1994-09

Automation of hardware-in-the-loop testing of control systems for unmanned air vehicles

Moats, Michael L.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

AUTOMATION OF HARDWARE-IN-THE-LOOP TESTING OF CONTROL SYSTEMS FOR UNMANNED AIR VEHICLES

by

Michael L. Moats

September, 1994

Thesis Advisor: Isaac I. Kaminer

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	September, 1994	Master's Thesis

4. TITLE AND SUBTITLE
AUTOMATION OF HARDWARE-IN-THE-LOOP TESTING OF CONTROL SYSTEMS FOR UNMANNED AIR VEHICLES

5. FUNDING NUMBERS

6. AUTHOR(S)
Michael L. Moats

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Modern computing advances allow the aerospace controls engineer the ability to design, test, and implement automatic control systems for air vehicles with breath taking speed and accuracy. This work examines the automation of the hardware-in-the-loop testing and implementaion of autonomous controllers for Unmanned Air Vehicles. Extraordinary interest is generated in this subject considering automation results in hardware-in-the-loop testing within days of completing a controller design. The entire automation process is presented, from design of the controller to implementation on a particular control platform to hardware-in-the-loop testing of the controller. This accomplishes control design and implemention in a matter of months compared to a few years or more before automation.

14. SUBJECT TERMS	NUMBER OF PAGES
Hardware–In–The–Loop Simulation, HITL, H Infinity Control, Real–Time Simulation, AROD, Pulse Width Modulation, PWM, AC100, MATRIXx, SystemBuild	109
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. Z39-18
298-102

Approved for public release; distribution is unlimited

AUTOMATION OF HARDWARE–IN–THE–LOOP TESTING OF CONTROL SYSTEMS FOR UNMANNED AIR VEHICLES

by

Michael L. Moats
Lieutenant, United States Navy
B.S. University of Colorado, Colorado Springs, 1986

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN AERONAUTICAL ENGINEERING

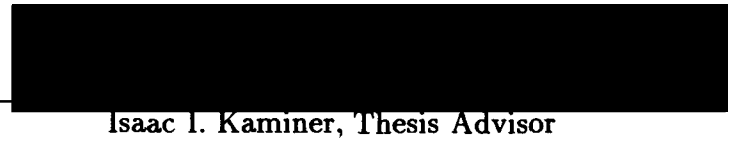
from the

NAVAL POSTGRADUATE SCHOOL

September , 1994

Author: 

Michael L. Moats

Approved by: 

Isaac I. Kaminer, Thesis Advisor

Daniel J. Collins, Second Reader

Daniel J. Collins, Chairman
Department of Aeronautics and Astronautics

ABSTRACT

Modern computing advances allow the aerospace controls engineer the ability to design, test, and implement automatic control systems for air vehicles with breath taking speed and accuracy. This work examines the automation of the hardware-in-the-loop testing and implementaic· ·· ·· ·nomous controllers for Unmanned Air Vehicles. Extraordinary interest is genera· ·· in this subject considering automation results in hardware-in-the-loop testing within days of completing a controller design. The entire automation process is presented, from desig· ·f the controller to implementation on a particular control platform to hardware-in-the-loop testing of the controller. This accomplishes control design and implemention in a mat·er of months compared to a few years or more before automation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGMENT

I would like to thank the many people who contributed to this thesis in their own individual ways. Dr. I. I. Kaminer for his guidance, teaching, and forethought in developing an outstanding avionics laboratory. Dr. D. J. Collins and Dr. R. M. Howard for their professional counsel. CDR. Duym for his consideration and genuine concern for my career. Ken Reyneveld and Paul Schmidt at Integrated Systems, Inc. for their technical support in answering seemingly millions of questions. Most of all, I would like to thank my wife Julie for understanding the least while sacrificing the most.

I. INTRODUCTION

The aeronautical controls engineer of the 90's can design a dynamic aircraft model, verify its accuracy, design a control system, and implement the controller on a computer in a matter of a few months. Application software tools such as MATLAB with SIMULINK and MATRIX$_X$ with SystemBuild allow the design to be completed and tested at the block diagram level. In particular, Autogen and AC100 products developed by Integrated Systems, Incorporated allow for direct conversion of the block diagram to a fully implemented control system. This system can be tested real-time with hardware-in-the-loop while recording any or all of the state variables to verify performance. Before discussing the automation of the design process in detail, a brief outline of each step follows.

The first step in the design process is to create a high fidelity nonlinear model of the aircraft which can be reliably trimmed and linearized throughout the full spectrum of required flight conditions. Such modeling is completed in four stages:

- Developing the equations of motion. Sum all of the forces and moments involved and write the equations in terms of states to allow for easy convertion into a block diagram.

- Creating a nonlinear model. Create a block diagram representation of these equations.

- Creating a linear model. Trim the nonlinear model about the desired trim point and then linearize the model to obtain a linear model.

- Validating the model. Use the data from an independent source to validate the model.

The next step is to design a feedback controller. This is typically an iterative procedure beginning with the design requirements. Usually requirements will be placed by the designer in terms of response time and overshoot for the desired controller. For example a heading controller may be required to achieve a ten degree heading change within 30 seconds and have a maximum overshoot of one degree. With the requirements on hand, the control design steps are:

- Building a control synthesis model. The designer chooses which sensors to use and creates a synthesis model with the desired command inputs using the linear model.

- Computing the control gain. A cost function is defined by the designer depending on the specified requirements and the method chosen for computing the controller. The control gain is then calculated. The methods for computing this gain include:

  - Linear Quadratic Regulator Theory

  - H_∞ Theory

- Check the command and control loop bandwidth: The bandwidth is plotted for each of the command loops and checked against the specified requirements. The control loop bandwidth is also checked and compared to the bandwidth of the chosen actuators.

The cost function defined above includes some weighting factors. These are the design knobs which the designer uses to create the desired controller. If the control

and/or command bandwidths are too large or too small, the designer adjusts these weights and re-computes the control gain. This typically involves many iterations. The designer may also find it necessary to move the zeros placed in the synthesis model and restart the iterations from that point.

The next step in the design process is to test the feedback system. SIMULINK and SystemBuild have built in capabilities for performing these tests. The first test is to close the loop with the linear model and the controller. The resulting closed-loop system is then tested. Next the controller is connected with the nonlinear model and tested.

An important step in the testing process is to develop an accurate model of the actuators. If the actuators are not modeled well, the software test of the controller may indicate that the controller works as designed while an hardware-in-the-loop test may show that the feedback system is unstable. This is usually caused by the actuators not being able to respond fast enough to commands.

Once an accurate model of the actuators has been developed, the closed loop software test is repeated with the actuator models in the loop. If the actuator models are acurate and the controller design is correct, there should not be an apparent difference in the performance of the controller.

The designed controller must be implemented on a platform capable of producing the required control signals and reading the given sensor outputs. Since personal computers, or PCs have become very cost effective, the typical controller implementation is a PC microprocessor with input/output, or I/O cards. The I/O cards available can produce or read analog voltages, pulse width modulated, or PWM signals, and serial communications to name a few. The control algorithm is then programmed using a high level language such as C or FORTRAN and then compiled to run on

the chosen PC. During programming, careful consideration must be given to initializing and using the I/O cards. Timing is the most critical part of the programmed controller.

The next step is hardware-in-the-loop , or HITL simulation, where the feedback system is tested with some of the actual hardware which will be used to control the aircraft. HITL testing is done in one or more stages.

- Actuators and Sensors: The first stage is to include all of the actuators which receive control signals directly, such as the elevators, rudder, and ailerons. The sensors which measure the results of these actuators must also be included in order to close the loop. After this initial test, other less critical actuators may be added.

- Control Inputs and Sensors: A possible second stage is to include the control input device, such as a joystick for an unmanned aircraft, into the loop along with the sensors required to measure the control inputs.

The most critical part of any hardware-in-the-loop test is calibration of the sensors. The controller includes an algebraic conversion of the measured sensor outputs to a signal that can be used directly by the controller. This algebraic conversion requires calibration by determining the correct conversion constants.

The ultimate test of a designed controller is the flight test. Budget considerations demand that the controller work perfectly the first time. The flight test phase is usually performed in three stages:

- Test Stand: The first flight test is usually done in a laboratory facility on a test stand which allows some degree of movement while restricting flight so that the vehicle can not be damaged.

- Tethered Flight: The next stage is typically a teathered flight. In this manner, an experienced pilot can be standing by with a manual override switch to allow direct manual control of the vehicle in the event of a problem.

- Actual Flight: The final and ultimate test of the control design is the autonomous flight test.

The main purpose of this report is to discuss the automation of the design process which has just been summarized. The main tool used in this process is Integrated Systems, Incorporated's AC100 package. Once the designer develops a plant model and a controller using MATRIX$_X$ and SystemBuild, AC100 can be used to implement and test the controller on actual hardware with a few pushes of a button.

II. DEVELOPING EQUATIONS OF MOTION

A. BACKGROUND

There are several unmanned air vehicles, or UAV's, currently in use a S. Two of these are discussed in this report, the Bluebird and the AROD. The AROD is described in the next section and the detailed development of its equations of motion and computer modeling are covered. The Bluebird is discussed next. It is a small conventional aircraft acquired as a test bed for testing guidance and navigation systems. For a complete description and the development of the equations of motion for Bluebird, see [Ref. 1].

B. DESCRIPTION OF AROD

The Airborne Remotely Operated Device, AROD is a vertical take-off and landing, VTOL, aircraft originally designed by Sandia Research Laboratory in Albuquerque, New Mexico. The AROD has been the subject of several theses at NPS and this report expands on the work started by those individuals. For a detailed description of AROD refer to [Ref. 2, 3] and the Sandia Lab papers [Ref. 4, 5] as well as the references therein. The AROD is shown in Figure 2.1 and its characteristics are tabulated in Table 2.1.

A combination of the control vanes are used to exert the desired control forces on the AROD. Roll control is obtained by deflecting all four vanes in the opposite direction of the desired roll. Pitch and yaw are obtained by deflecting the pair of vanes in the y and z planes respectively. The numbering of the vanes is shown in

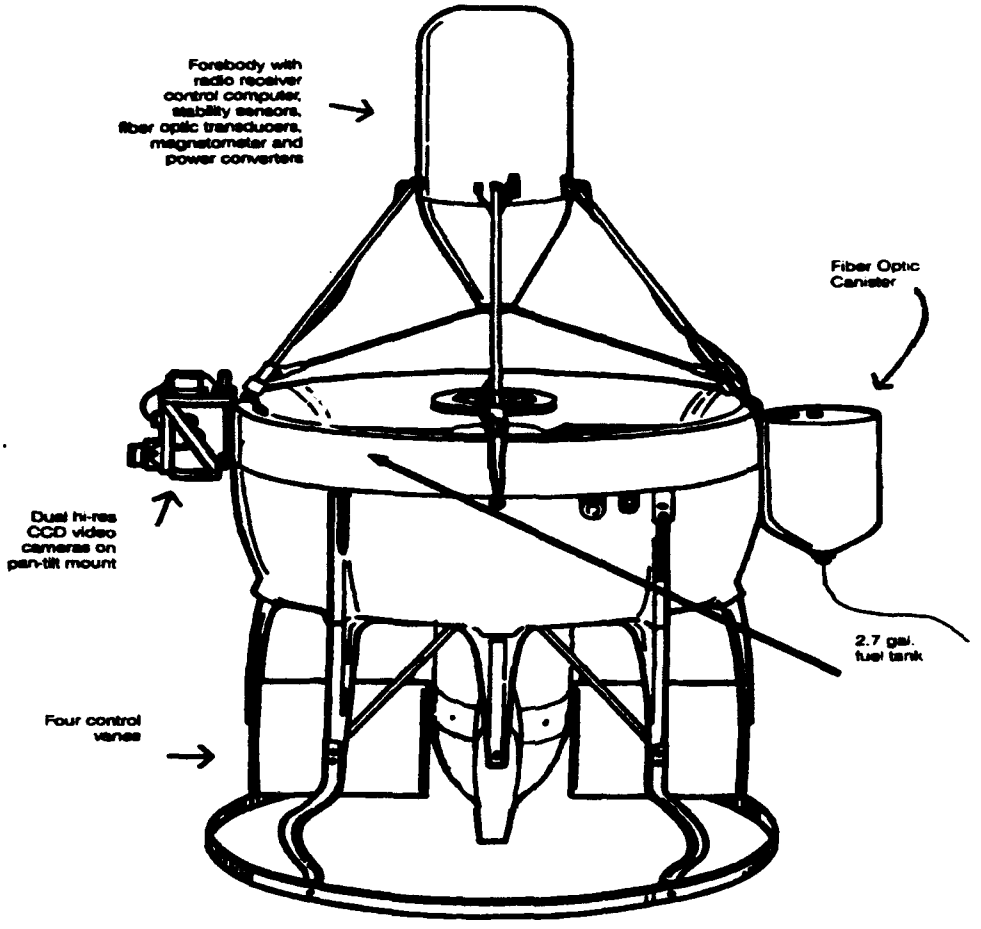


Figure 2.1: Airborne Remotely Operated Device, AROD

Figure 2.2 and the combinations of vanes required for roll, pitch and yaw are given in Table 2.2.

There are three dynamic coupling effects which must be considered when designing a control system for AROD. The first is the gyroscopic coupling due to the large angular momentum of the propeller. Another dynamic coupling exists between the vehicle attitude and the altitude-rate since thrust vectoring is required for translational movement. This coupling is depicted in Figure 2.3. The third effect is caused by the propeller. As the air is accelerated through the propeller, it is also deflected

TABLE 2.1: PHYSICAL CHARACTERISTICS OF AROD

Inlet Diameter, A	29.25 in
Propeller Radius, R	12 in
Exit Radius	23.375 in
Inlet Area Ratio	1.219
Exit Area Ratio	1.115
Exterior Contour	Tapered Rear
Propeller Location, % chord	25 %
Number of Blades	3
Engine Speed, Max.	8000 rpm
Engine Speed, Nom.	6500 rpm
Tip Speed, Max.	838 fpm
Tip Speed, Nom.	680 fpm
Power Loading, $\frac{BHP(\rho_0/\rho)}{R^2/4}$	7.25 HP/f^2
Mass Moment of Inertia, I_x	1.8241 $slug - f^2$
Mass Moment of Inertia, I_y	1.7997 $slug - f^2$
Mass Moment of Inertia, I_z	1.6147 $slug - f^2$
Prop Mass Moment of Inertia, I_{rx}	0.0311 $slug - f^2$
Prop Mass Moment of Inertia, I_{ry}	0.0067 $slug - f^2$
Prop Mass Moment of Inertia, I_{rz}	0.0067 $slug - f^2$

TABLE 2.2: VANE DEFLECTION COMBINATIONS FOR POSITIVE ANGLES

	Vane Combination
Roll, Φ	$V_1 + V_2 + V_3 + V_4$
Pitch, Θ	$V_2 - V_4$
Yaw, Ψ	$V_1 - V_3$

slightly causing a swirl effect. The result is that the air mass strikes the control vanes at an angle proportional to the angular rate of the propeller. This creates a rolling moment which is dependent on the throttle input.

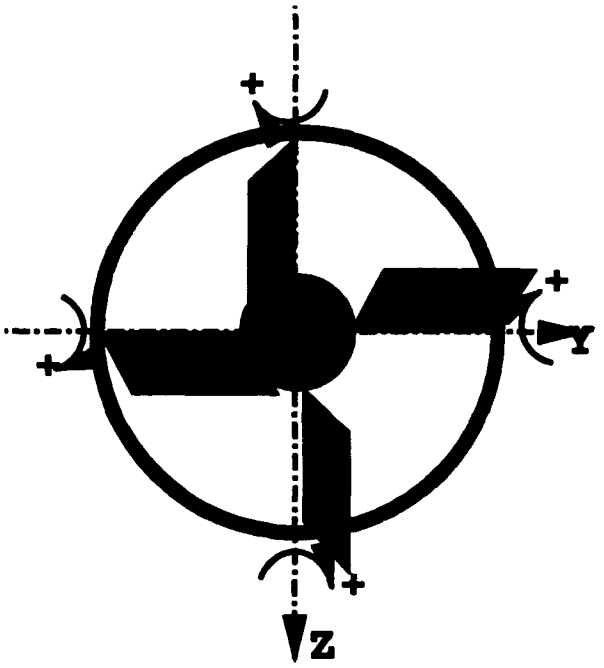


Figure 2.2: AROD Direction of Positive Vane Deflections

C. EQUATIONS OF MOTION

1. Notation

The notation used in this report is consistent with the previous work on the AROD, see [Ref. 2] and references therein. Consider Figure 2.4, here:

- $\{A\}$ represents the coordinate system with basis vectors, x_A, y_A, and z_A.
- ${}^{A}P_Q$ represents the position of point Q, expressed in $\{A\}$.
- ${}^{A}V_Q$ represents the velocity of point Q, measured in $\{A\}$ and expressed in $\{A\}$.
- ${}^{B}({}^{A}V_Q)$ represents the velocity of point Q, measured in $\{A\}$, and expressed in $\{B\}$.
- ${}^{A}_{B}R$ is a rotation matrix from $\{B\}$ to $\{A\}$, also called a direction cosine matrix. A free vector in one coordinate system, is a vector that *can be moved parallel*

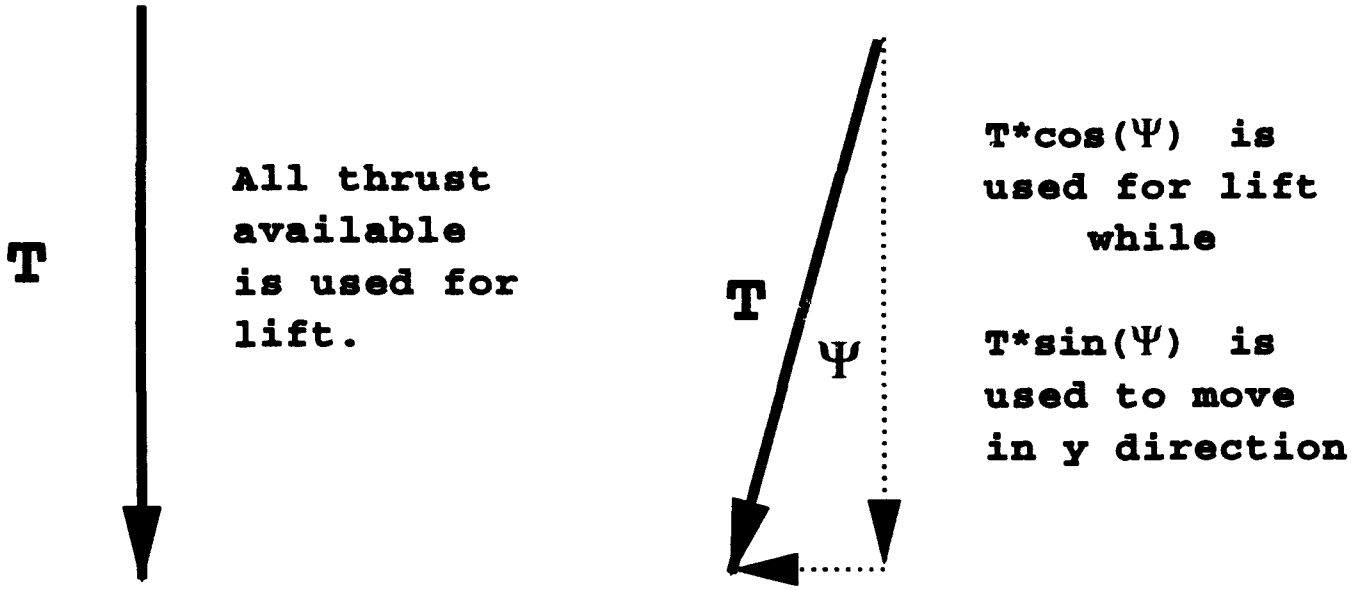


Figure 2.3: Coupling Between Altitude and Attitude

to itself without change. As a result, it can be expressed in another coordinate system by using the rotation matrix. For example:

$$^{A}(^{B}V_{Q}) = {}^{A}_{B}R\,(^{B}V_{Q})$$

- ${}^{A}\Omega_{B}$ is the angular velocity of the $\{B\}$ coordinate system with respect to $\{A\}$, and expressed in $\{A\}$.
- ${}^{B}({}^{A}\Omega_{B})$ is the angular velocity of $\{B\}$, with respect to $\{A\}$, and expressed in $\{B\}$.
- Additional information can be added to the subscripts i.e., ${}^{A}P_{BO}$ is the position of the origin of $\{B\}$, expressed in $\{A\}$.

2. Coordinate Systems

In order to derive equations of motion for a rigid airplane, the following coordinate systems and assumptions will be used:

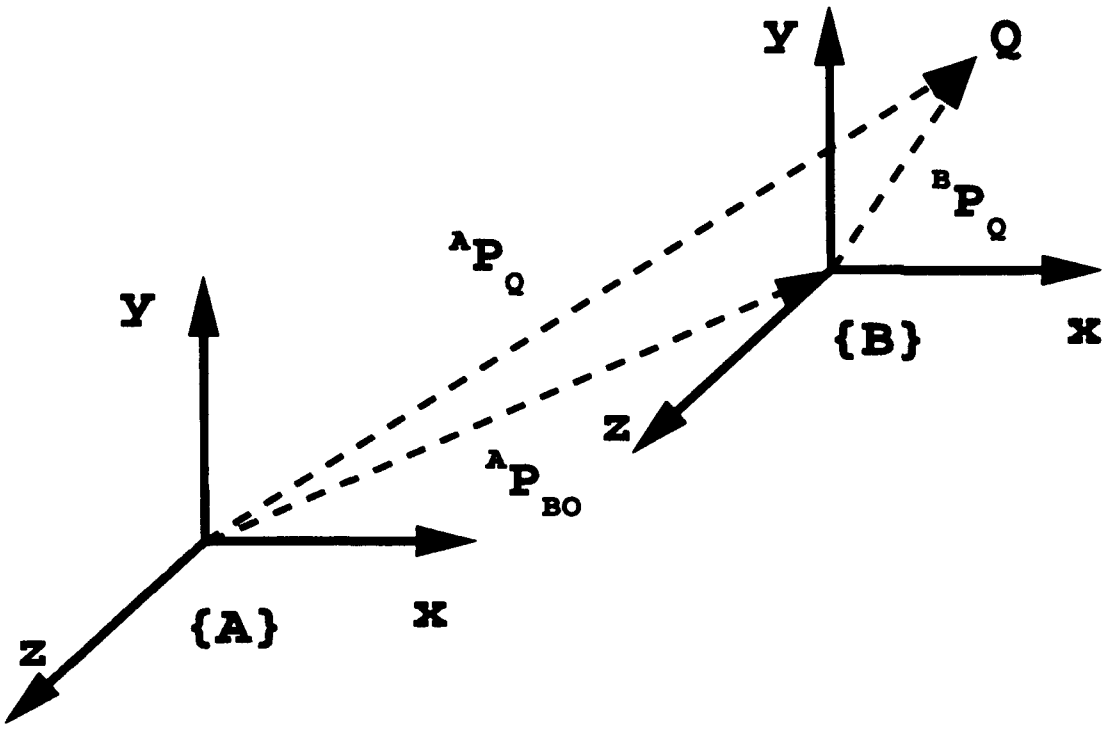


Figure 2.4: Relative Position of Coordinate Systems

- $\{U\}$ represents the inertial tangent plane coordinate system attached to Earth.
- $\{B\}$ represents the body fixed coordinate system.
- $\{W\}$ represents the wind axis coordinate system.
- All sensors are located at the c.g. (This assumption is used for simplification only and can be relaxed as shown in [Ref. 2])
- The mass of the aircraft remains constant.
- Given a vector v, its derivative with respect to $\{B\}$ is denoted as $\frac{d}{dt}(v)$ and its derivative with respect to $\{U\}$ is denoted as $(\dot{v})$

The $\{B\}$ coordinate system is a right handed system with X_B defined as the thrust axis. A positive roll rate, p, is clockwise when looking in the positive X direction.

The positive direction for Y_B, the pitch axis, is out the right wing . A positive pitch rate, q, is defined as clockwise looking in the positive Y direction. The Z_B axis is the yaw axis, and a positive yaw rate, r, is defined as clockwise, looking in the positive Z direction.

The $\{W\}$ coordinate system is defined with X_W aligned with the wind incident on the aircraft. The angle α is the angle formed by the body x-y plane and the positive X_W axis. The angle β is the angle formed by the body x-z plane and the positive Y_W axis.

To simplify the notation in places where it becomes cumbersome, the following definitions are introduced:

- v_Q represents the velocity of an arbitrary point, Q, measured and expressed in $\{U\}$.
- v_{BO} represents the velocity of the origin of $\{B\}$, measured and expressed in $\{U\}$, i.e., ${}^U V_{BO} = v_{BO}$.
- $\dot{v}_B$ represents the acceleration of $\{B\}$ with respect to $\{U\}$, measured and expressed in $\{U\}$.
- ${}^B v_Q$ represents the velocity of point Q, measured in $\{U\}$ and expressed in $\{B\}$, i.e., ${}^B({}^U V_Q) = {}^B v_Q$.
- ω_B represents the angular velocity of $\{B\}$, measured and expressed in $\{U\}$, i.e., ${}^U\Omega_B = \omega_B$.
- ${}^B\omega_B$ represents the angular velocity of $\{B\}$ measured in $\{U\}$, and expressed in $\{B\}$, i.e., ${}^B({}^U\Omega_B) = {}^B\omega_B$.
- 0 represents the appropriate size matrix with all elements equal to zero.

- I_n represents the identity matrix of dimension n.

3. Spatial Orientation Using Euler Angles

The spatial orientation of a rigid body [Ref. 6] can be defined by the three Euler angles, Φ, Θ, and Ψ called roll, pitch and yaw and defined in Figure 2.5. The Euler angles, in turn, can be used to define a rotation between two coordinate systems. This rotation is obtained using Euler's theorem:

> Any number of rotations about different axes through a point must, in the end, remain equivalent to a single rotation.

For the case of conventional aircraft, a 3–2–1 rotation sequence is used [Ref. 7], where the aircraft is yawed, pitched, and then rolled. In this case, Θ is small, and in steady state flight is equal to the angle of attack, α. The angle Φ can be expected to be anywhere from $\pm$ 60 deg in normal flight and can be anywhere from $\pm$ 180 deg in acrobatic flight. Ψ represents the heading of the aircraft and of course can range from 0 to 360 deg. This euler angle rotation was used in modeling the Bluebird.

Euler angle rotations have an inherent singularity point when considering euler angular rates. The singularity point for a 3–2–1 rotation is $\Theta = 90$ deg. Therefore, the adopted convention for AROD is a 2–3–1 rotation which has a singularity point at $\Psi = 90$ deg.

The transformation from inertial coordinates$\{U\}$, to body coordinates $\{B\}$, is carried out as follows, and is shown in Figure 2.5.

1. The inertial coordinate system is represented by the vector ${}^{U}V$, with the components x, y, and z. The first rotation is made about the y axis through an angle Θ. Now the vector is expressed as ${}^{2}V$ with the components x_2, y_2, and z_2.

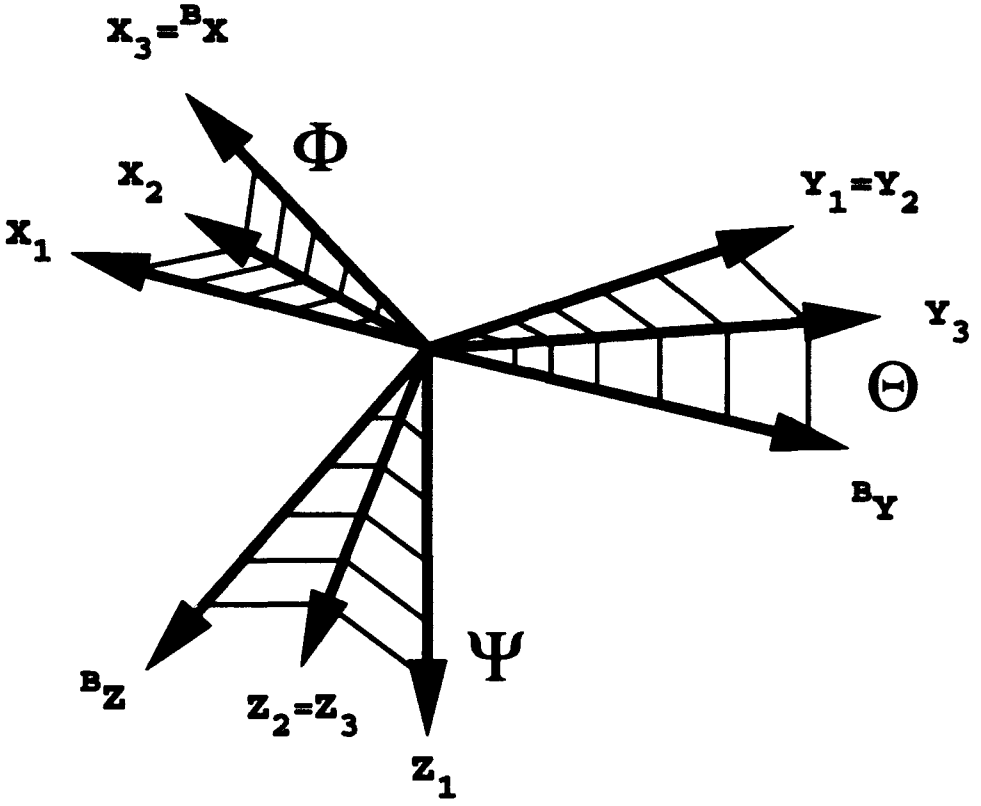


Figure 2.5: Y–Z–X Euler Angle Rotation Sequence

Since the rotation was about the y axis, the y_2 component remains unchanged. The resulting elemental matrix is:

$$M(\Theta) = \begin{bmatrix} \cos\Theta & 0 & -\sin\Theta \\ 0 & 1 & 0 \\ \sin\Theta & 0 & \cos\Theta \end{bmatrix}. \tag{2.1}$$

2. Now the rotation is made about the new z axis, z_2, through an angle Ψ. This results in a third coordinate system with the vector expressed as ${}^3V$, and having components x_3, y_3, and z_3. This rotation does not change the z_3 component. The resulting elemental matrix is:

$$M(\Psi) = \begin{bmatrix} \cos\Psi & \sin\Psi & 0 \\ -\sin\Psi & \cos\Psi & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.2}$$

3. Lastly, the rotation about the x_3 axis through an angle Φ is made to give the vector expressed in body coordinates, ${}^{B}V$. Now the resulting elemental matrix is:

$$M(\Phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\Phi & \sin\Phi \\ 0 & -\sin\Phi & \cos\Phi \end{bmatrix}. \tag{2.3}$$

When the matrices are multiplied together in the correct sequence, $M(\Phi)M(\Psi)M(\Theta)$, the result is the ${}^{B}_{U}R$ direction cosine matrix, expressed in terms of Euler angles as:

$$\begin{bmatrix} \cos\Psi\cos\Theta & \sin\Psi & -\cos\Psi\sin\Theta \\ -\cos\Phi\sin\Psi\cos\Theta + \sin\Phi\cos\Theta & \cos\Phi\cos\Psi & \cos\Phi\sin\Psi\cos\Theta + \sin\Phi\cos\Theta \\ \sin\Phi\sin\Psi\cos\Theta + \cos\Phi\sin\Theta & -\sin\Phi\cos\Psi & -\sin\Phi\sin\Psi\sin\Theta + \cos\Phi\cos\Theta \end{bmatrix} \tag{2.4}$$

The next step is to develop the kinematic differential equations that describe the change in Euler angles with time. Following the method used in [Ref. 7], the matrix of differential equations, Ω, can be written as a sum of individual Euler angle rates:

$$\Omega = M(\Phi)\begin{bmatrix} \frac{d}{dt}\Phi \\ 0 \\ 0 \end{bmatrix} + M(\Phi)M(\Psi)\begin{bmatrix} 0 \\ 0 \\ \frac{d}{dt}\Psi \end{bmatrix} + M(\Phi)M(\Psi)M(\Theta)\begin{bmatrix} 0 \\ \frac{d}{dt}\Theta \\ 0 \end{bmatrix}. \tag{2.5}$$

When the matrix algebra in Equation 2.5 is done, the resulting kinematic differential equations for p, q, and r are given as:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & \sin\Psi & 0 \\ 0 & \cos\Phi & \cos\Psi\sin\Phi \\ 0 & -\sin\Phi & \cos\Psi\cos\Phi \end{bmatrix}\begin{bmatrix} \dot{\Phi} \\ \dot{\Theta} \\ \dot{\Psi} \end{bmatrix} \tag{2.6}$$

The matrix on the right hand side of Equation 2.6 is invertible for all $\Psi \neq \frac{\pi}{2}$, and can be used to solve for the Euler angle rates, $\dot{\Phi}, \dot{\Theta}$ and $\dot{\Psi}$:

$$\begin{bmatrix} \dot{\Phi} \\ \dot{\Theta} \\ \dot{\Psi} \end{bmatrix} = \begin{bmatrix} 1 & -\cos\Phi\tan\Psi & \sin\Phi\tan\Psi \\ 0 & \cos\Phi\sec\Psi & -\sin\Phi\sec\Psi \\ 0 & \sin\Phi & \cos\Phi \end{bmatrix}\begin{bmatrix} p \\ q \\ r \end{bmatrix}. \tag{2.7}$$

The time history of the Euler angles can be obtained by integrating Equation 2.7.

4. Derivation of the Equations of Motion

For a general aircraft model with six degrees of freedom the derivation of the equations of motion can be broken into two parts. The first part is the motion of an arbitrary rigid body in free space. This motion depends only on the linear and angular momenta of the rigid body which can be divided into linear and angular equations. The second part is an examination of all of the forces acting on that rigid body. These forces are aerodynamic, gravitational, and propulsive. The aerodynamic and propulsive forces are specific to the aircraft being modeled and are characterized by the stability and control derivatives described later in this thesis.

a. Linear Equations

The linear equations are developed using Newton's Law, $F = ma$. Because the sensors are attached to the body of the aircraft, the equations are written in the $\{B\}$ coordinate system. Matrix equations avoid the repetition of writing equations in terms of x, y, and z. First the position of the aircraft center of gravity, or c.g., (the origin of $\{B\}$) is determined as ${}^{U}P_{BO}$. Next Coriolis' theorem is applied to obtain linear velocities for the aircraft. Coriolis' theorem is then applied a second time to derive the equation for linear accelerations. First, define:

$$ {}^{U}V_{BO} \triangleq {}^{U}\dot{P}_{BO}. \tag{2.8} $$

Both sides of Equation 2.8 are premultiplied by ${}^{B}_{U}R$ to get:

$$ {}^{B}_{U}R\,{}^{U}V_{BO} = {}^{B}_{U}R\,{}^{U}\dot{P}_{BO} $$

or

$$ {}^{B}v_{BO} = {}^{B}\dot{P}_{BO} \tag{2.9} $$

Now consider Coriolis' theorem:

$$\dot{A} = \frac{d}{dt}A + \omega \times A, \tag{2.10}$$

where $\dot{A}$ and $\frac{d}{dt}A$ use the notation for derivatives previously defined in Section 1 The term $\omega \times A$ represents the difference between the relative velocity as measured from the rotating and non-rotating axes [Ref. 8].

Equation 2.10 is applied to ${}^{B}v_B$ in Equation 2.9 to get:

$${}^{B}\dot{v}_{BO} = \frac{d}{dt}{}^{B} v_B + {}^{B}\omega_B \times {}^{B} v_B, \tag{2.11}$$

Newton's law can now be written as:

$$\begin{aligned} {}^{U}F &= m\,{}^{U}a \\ &= m\,\dot{v}_{BO}, \end{aligned} \tag{2.12}$$

where ${}^{U}F$ is the total external force applied to the aircraft c.g. Equation 2.12 is premultiplied by ${}^{B}_{U}R$ to obtain the result:

$$\begin{aligned} {}^{B}F &= m\,{}^{B}_{U}R\,{}^{U}\dot{v}_{BO} \\ &= m\,{}^{B}\dot{v}_{BO}. \end{aligned} \tag{2.13}$$

when Equation 2.11 is substituted into Equation 2.13, the final result for ${}^{B}F$ is:

$$\begin{aligned} {}^{B}F &= m\,(\frac{d}{dt}{}^{B} v_B + {}^{B}\omega_B \times {}^{B} v_B) \\ &= m\,\frac{d}{dt}{}^{B} v_B + m\,{}^{B}\omega_B \times {}^{B} v_B. \end{aligned} \tag{2.14}$$

b. Angular Equations

The angular equations are derived using Euler's Law for preservation of angular momentum. These equations are derived in the $\{B\}$ coordinate system for

the c.g. by applying Coriolis' theorem to the equation for Euler's Law:

$$^{U}\dot{L}_{BO} = {}^{U}N_{BO}, \tag{2.15}$$

where $^{U}L_{BO}$ is the angular momentum of the aircraft and $^{U}N_{BO}$ is the total external moment applied to the aircraft c.g. Then, premultiplying Equation 2.15 by ${}^{B}_{U}R$ gives:

$$^{B}\dot{L}_{BO} = {}^{B}_{U}R\,{}^{U}N_{BO}, \tag{2.16}$$

Using Coriolis' theorem in Equation 2.10, $^{B}\dot{L}_{BO}$ can be rewritten as:

$$^{B}\dot{L}_{BO} = \frac{d}{dt}{}^{B}L_{BO} + {}^{B}\omega_{B} \times {}^{B}L_{BO}, \tag{2.17}$$

The angular momentum, $^{B}L_{BO}$, is defined as the sum of the product of the inertia tensor, I_B, and the body's angular velocity, $^{B}\omega_{B}$, and the product of the inertia tensor I_R, and the angular velocity of any rotating parts $^{B}\omega_{R}$, or:

$$^{B}L \triangleq I_B\,{}^{B}\omega_{B} + I_R\,{}^{B}\omega_{R}, \tag{2.18}$$

where I_R and $^{B}\omega_{R}$ are the moment of inertia and the angular velocity of the rotating part, respectively. Note that additional rotating parts can be accounted for by adding additional terms to Equation 2.18. With a single propeller the equation becomes:

$$^{B}L \triangleq I_B\,{}^{B}\omega_{B} + I_P\,({}^{B}\omega_{B} + {}^{B}\omega_{P}), \tag{2.19}$$

where I_P and $^{B}\omega_{P}$ are the moment of inertia and the angular velocity of the propeller, respectively. When this term is substituted into Equation 2.17, the result is:

$$^{B}\dot{L}_{BO} = \frac{d}{dt}(I_B{}^{B}\omega_{B} + I_P({}^{B}\omega_{B} + {}^{B}\omega_{P})) + {}^{B}\omega_{B} \times (I_B{}^{B}\omega_{B} + I_P({}^{B}\omega_{B} + {}^{B}\omega_{P})), \tag{2.20}$$

For simplification, define the total inertia tensor, I_T as:

$$I_T \triangleq I_B + I_P \tag{2.21}$$

Collecting terms, Equation 2.20 becomes:

$$^{B}\dot{L}_{BO} = \frac{d}{dt}(I_T{}^{B}\omega_B + I_P{}^{B}\omega_P) + {}^{B}\omega_B \times (I_T{}^{B}\omega_B + I_P{}^{B}\omega_P), \tag{2.22}$$

Carrying out the differentiation in Equation 2.22 yields:

$$^{B}\dot{L}_{BO} = I_T\frac{d}{dt}{}^{B}\omega_B + I_P\frac{d}{dt}{}^{B}\omega_P + {}^{B}\omega_B \times (I_T{}^{B}\omega_B + I_P{}^{B}\omega_P). \tag{2.23}$$

Since $\frac{d}{dt}({}^{B}\omega_B) = {}^{B}\dot{\omega}_B$ and $\frac{d}{dt}({}^{B}\omega_R) = 0$ if we assume a constant angular velocity for the propeller, Equation 2.23 can be simplified to:

$$^{B}\dot{L}_{BO} = I_T{}^{B}\dot{\omega}_B + {}^{B}\omega_B \times (I_T{}^{B}\omega_B + I_P{}^{B}\omega_P) \tag{2.24}$$

Now the result in Equation 2.24 can be substituted into Equation 2.16:

$$^{B}N_{BO} = I_T{}^{B}\dot{\omega}_B + {}^{B}\omega_B \times (I_T{}^{B}\omega_B + I_P{}^{B}\omega_P). \tag{2.25}$$

The term $I_P{}^{B}\omega_P$ can be disregarded if it is insignificant compared to I_B and ${}^{B}\omega_B$ [Ref. 9]. This term is neglected in modeling the Bluebird see [Ref. 1]. For the case of AROD this term is significant and is not neglected.

c. State Equations

Now that the kinematic equations of motion have been developed in matrix form, these equations can be assembled into a state-space representation of the equations of motion. First, Equations 2.14 and 2.25 can be written as:

$$\begin{bmatrix} {}^{B}F \\ {}^{B}N \end{bmatrix} = \begin{bmatrix} m\,\frac{d}{dt}{}^{B}v_B + m\,({}^{B}\omega_B \times {}^{B}v_B) \\ I_T{}^{B}\dot{\omega}_B + {}^{B}\omega_B \times (I_T{}^{B}\omega_B + I_P{}^{B}\omega_P) \end{bmatrix}. \tag{2.26}$$

Equation 2.26 can be rearranged to yield:

$$\frac{d}{dt}\begin{bmatrix} m\,{}^{B}v_B \\ I_T{}^{B}\omega_B \end{bmatrix} = \begin{bmatrix} m\,(-{}^{B}\omega_B \times {}^{B}v_B) & +{}^{B}F) \\ -{}^{B}\omega_B \times (I_T{}^{B}\omega_B + I_P{}^{B}\omega_P) & +{}^{B}N) \end{bmatrix}. \tag{2.27}$$

The terms on the left hand side of Equation 2.27 can be normalized by multiplying by $\frac{1}{m}$ and I_T^{-1}, with the final result:

$$\frac{d}{dt}\begin{bmatrix} {}^B v_B \\ {}^B \omega_B \end{bmatrix} = \begin{bmatrix} -{}^B\omega_B \times {}^B v_B & + & \frac{{}^B F}{m} \\ -I_T^{-1}\,{}^B\omega_B \times (I_T {}^B\omega_B + I_P {}^B\omega_P) & + & I_T^{-1}\,{}^B N \end{bmatrix}. \quad (2.28)$$

d. Forces and Moments

Equation 2.28 gives the kinematic equations of motion for a rigid body. The next step is to examine the forces ${}^B F$ and moments ${}^B N$ acting on the rigid body. These forces and moments are due to gravity, aerodynamics, and propulsion, and are written as:

$$\begin{bmatrix} {}^B F \\ {}^B N \end{bmatrix} = \begin{bmatrix} {}^B F_{GRAV} + {}^B F_{PROP} + {}^B F_{AERO} \\ {}^B N_{PROP} + {}^B N_{AERO} \end{bmatrix} \quad (2.29)$$

Gravitational Forces: The gravitational forces acting on the aircraft, ${}^B F_{GRAV}$, can be determined by rotating ${}^U F_{GRAV}$ with the appropriate rotation matrix, where:

$${}^U F_{GRAV} = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix}. \quad (2.30)$$

Then:

$${}^B F_{GRAV} = {}^B_U R\, {}^U F_{GRAV}. \quad (2.31)$$

Aerodynamic Forces and Moments: The aerodynamic force and moment terms are determined by using first–order Taylor series expansion around a given nominal operating point. This operating point is the state chosen to represent the aircraft's flight condition. Each term in the series is a partial derivative of ${}^B F$ or ${}^B N$ with respect to the aerodynamic variables, $\frac{u}{U}$, α, β, p, q, r [Ref. 7, 10]:

$$F_{AERO} = \delta F_{x'} x' + \delta F_{\dot{x}'} \dot{x}' + \delta F_{\Delta} \Delta + F_0. \quad (2.32)$$

Similarly, moment terms can be written as:

$$N_{AERO} = \delta N_{x'} x' + \delta N_{\dot{x}'} \dot{x}' + \delta N_{\Delta} \Delta + N_0, \tag{2.33}$$

where x' is given by:

$$x' = \begin{bmatrix} \frac{u}{U} \\ \beta \\ \alpha \\ \frac{pb}{2U} \\ \frac{qc}{2U} \\ \frac{rb}{2U} \end{bmatrix}, \tag{2.34}$$

and $\dot{x}'$ is given as:

$$\dot{x}' = \begin{bmatrix} \dot{\beta} \\ \dot{\alpha} \end{bmatrix}. \tag{2.35}$$

Notice that $\dot{x}'$ contains only two elements. The other derivatives are negligible and therefore not included. Control inputs are represented by the vector Δ:

$$\Delta = \begin{bmatrix} \delta_e \\ \delta_r \\ \delta_a \end{bmatrix} \tag{2.36}$$

where δ_e, δ_r, and δ_a are the elevator, rudder, and aileron inputs, respectively. Equations 2.32–2.36 can now be combined as follows:

$$\begin{bmatrix} {}^{W}F_{AERO} \\ {}^{W}N_{AERO} \end{bmatrix} = \bar{q}\bar{S}\{\frac{\partial C}{\partial x'} x' + \frac{\partial C}{\partial \dot{x}'} \dot{x}' + \frac{\partial C}{\partial \Delta} \Delta + C_{FO}\}, \tag{2.37}$$

where $\bar{q} = \frac{1}{2}\rho V^2$, $\bar{S} = \text{diag}\{S, S, S, Sb, Sc, Sb\}$, and C is the matrix of non–dimensional stability derivatives differentiated with respect to the terms defined in Equation 2.34, 2.35, or 2.36. $\frac{\partial C}{\partial x'}$ is defined as:

$$\frac{\partial C}{\partial x'} \triangleq \begin{bmatrix} C_{L_U} & C_{L_\beta} & C_{L_\alpha} & C_{L_p} & C_{L_q} & C_{L_r} \\ C_{Y_U} & C_{Y_\beta} & C_{Y_\alpha} & C_{Y_p} & C_{Y_q} & C_{Y_r} \\ C_{D_U} & C_{D_\beta} & C_{D_\alpha} & C_{D_p} & C_{D_q} & C_{D_r} \\ C_{l_U} & C_{l_\beta} & C_{l_\alpha} & C_{l_p} & C_{l_q} & C_{l_r} \\ C_{m_U} & C_{m_\beta} & C_{m_\alpha} & C_{m_p} & C_{m_q} & C_{m_r} \\ C_{n_U} & C_{n_\beta} & C_{n_\alpha} & C_{n_p} & C_{n_q} & C_{n_r} \end{bmatrix}. \tag{2.38}$$

$\frac{\partial C}{\partial \dot{x}^i}$ is very similar to $\frac{\partial C}{\partial x^i}$, except that only the $\dot{\alpha}$ and $\dot{\beta}$ terms are normally significant, leaving a 6×2 matrix rather than a square matrix. The term $\frac{\partial C}{\partial \Delta}$ is defined as:

$$\frac{\partial C}{\partial \Delta} \triangleq \begin{bmatrix} C_{L_{\delta_e}} & C_{L_{\delta_r}} & C_{L_{\delta_a}} \\ C_{Y_{\delta_e}} & C_{Y_{\delta_r}} & C_{Y_{\delta_a}} \\ C_{D_{\delta_e}} & C_{D_{\delta_r}} & C_{D_{\delta_a}} \\ C_{l_{\delta_e}} & C_{l_{\delta_r}} & C_{l_{\delta_a}} \\ C_{m_{\delta_e}} & C_{m_{\delta_r}} & C_{m_{\delta_a}} \\ C_{n_{\delta_e}} & C_{n_{\delta_r}} & C_{n_{\delta_a}} \end{bmatrix}. \tag{2.39}$$

C_{F0} is defined to be the vector of steady state coefficients:

$$C_{F0} = \begin{bmatrix} C_{D0} \\ C_{Y0} \\ C_{L0} \\ C_{l0} \\ C_{m0} \\ C_{n0} \end{bmatrix}, \tag{2.40}$$

representing conditions in trimmed, balanced flight. This definition is similar to the definition used by Roskam [Ref. 9]. In other references, the term C_{F0} can refer to the nominal value of the coefficient at $\alpha = 0$. However, in the Taylor series expansion it is more natural to use the first definition of C_{F0}; therefore, it will be used in the following derivation and modeling. The stability and control derivatives are usually computed in the wind axis coordinate system, $\{W\}$. The transformation from $\{W\}$ to $\{B\}$ is performed in the same fashion as the Euler angle transformations mentioned earlier. The rotation matrix, ${}^B_W R$, is a function of α and β, and is expressed as:

$${}^B_W R = \begin{bmatrix} \cos\alpha\cos\beta & -\cos\alpha\sin\beta & -\sin\alpha \\ \sin\beta & \cos\beta & 0 \\ \sin\alpha\cos\beta & -\sin\alpha\sin\beta & \cos\alpha \end{bmatrix} \tag{2.41}$$

The rotation from $\{W\}$ to $\{B\}$ is made by premultiplying the force or moment vector by ${}^B_W R$.

Since lift and drag are defined as positive along the negative z_B and x_B

axes, we define F_{AERO} and N_{AERO} as:

$$F_{AERO} = \begin{bmatrix} -D \\ Y \\ -L \end{bmatrix} \text{ and } N_{AERO} = \begin{bmatrix} l \\ m \\ n \end{bmatrix}. \tag{2.42}$$

The negative sign on D and L can be moved into the $\bar{S}$ matrix for convenience, so that the new matrix is:

$$\bar{S} = \text{diag}\{-S, S, -S, Sb, Sc, Sb\} \tag{2.43}$$

In order to write Equation 2.37 in state space form, state variables must be defined. The most commonly used notation to use for the state vector is to use:

$$x = \begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \end{bmatrix}. \tag{2.44}$$

However, the terms x' and $\dot{x}'$ in Equations 2.32 and 2.33 cannot be used directly as states. It is easy to define:

$$\begin{aligned} M' &: x \rightarrow x' \\ \dot{M}' &: \dot{x} \rightarrow \dot{x}' \end{aligned} \tag{2.45}$$

where:

$$M' = \text{diag}\{\frac{1}{V_T}, \frac{1}{V_T}, \frac{1}{V_T}, \frac{b}{2V_T}, \frac{c}{2V_T}, \frac{b}{2V_T}\} \tag{2.46}$$

and:

$$\dot{M}' = \begin{bmatrix} 0 & \frac{c}{2V_T^2} & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{b}{2V_T^2} & 0 & 0 & 0 \end{bmatrix} \tag{2.47}$$

are matrices of appropriate dimensions. The complete expression for ${}^{B}F_{AERO}$ and ${}^{B}N_{AERO}$ can now be written as:

$$\begin{bmatrix} {}^{B}F_{AERO} \\ {}^{B}N_{AERO} \end{bmatrix} = \bar{q}\bar{S} \begin{bmatrix} {}^{B}_{W}R & 0 \\ 0 & {}^{B}_{W}R \end{bmatrix} \{C_{F_0} + \frac{\partial C}{\partial x'} M' x + \frac{\partial C}{\partial \dot{x}'} \dot{M}' \dot{x} + \frac{\partial C}{\partial \Delta} \Delta\} \tag{2.48}$$

and can be substituted into Equation 2.28.

Propulsive Forces and Moments: The propulsive forces and moments, ${}^{B}F_{PROP}$ and ${}^{B}N_{PROP}$, are computed directly in the body coordinate system $\{B\}$ and are expressed as:

$$ {}^{B}F_{PROP} = \begin{bmatrix} T_X \\ T_Y \\ T_Z \end{bmatrix}, \tag{2.49} $$

and:

$$ {}^{B}N_{PROP} = \begin{bmatrix} T_l \\ T_m \\ T_n \end{bmatrix}, \tag{2.50} $$

where the T_i's represent the forces or moments due to thrust. Computation of propulsive forces and moments depends on each particular engine installation, and must be determined for the individual aircraft modeled. For the AROD, the thrust is aligned in the X_B direction and located at the c.g. yielding:

$$ {}^{B}F_{PROP} = \begin{bmatrix} T_X \\ 0 \\ 0 \end{bmatrix}, \tag{2.51} $$

and:

$$ {}^{B}N_{PROP} = \begin{bmatrix} T_l \\ 0 \\ 0 \end{bmatrix}, \tag{2.52} $$

Equations 2.31, 2.51, and 2.52 can now be substituted into Equation 2.28:

$$ \frac{d}{dt}\begin{bmatrix} {}^{B}v_{BO} \\ {}^{B}\omega_B \end{bmatrix} = \begin{bmatrix} -{}^{B}\omega_B\times & 0 \\ 0 & -{}^{B}I_T^{-1}({}^{B}\omega_B \times ({}^{B}I_T{}^{B}\omega_B + I_P{}^{B}\omega_P)) \end{bmatrix} \begin{bmatrix} {}^{B}v_{BO} \\ {}^{B}\omega_B \end{bmatrix} + \begin{bmatrix} \frac{I_3}{m} & 0 \\ 0 & {}^{B}I_T^{-1} \end{bmatrix} \begin{bmatrix} {}^{B}F \\ {}^{B}N \end{bmatrix}, \tag{2.53} $$

where:

$$ \begin{bmatrix} {}^{B}F \\ {}^{B}N \end{bmatrix} = \left\{ \begin{bmatrix} {}^{B}F_{GRAV} \\ 0 \end{bmatrix} + \begin{bmatrix} {}^{B}F_{PROP} \\ {}^{B}N_{PROP} \end{bmatrix} \delta_T + \left\{ \begin{bmatrix} {}^{B}_{W}R & 0 \\ 0 & {}^{B}_{W}R \end{bmatrix} \cdot \bar{q}\bar{S} \left\{ C_{FO} + \frac{\partial C}{\partial x'} M'x + \frac{\partial C}{\partial \dot{x}'} \dot{M}'\dot{x} + \frac{\partial C}{\partial \Delta}\Delta \right\} \right\} \right\}. \tag{2.54} $$

e. Complete Equations of Motion

In order to write Equation 2.28 in state space form, the terms associated with $\dot{x}'$ must be collected and moved to the left hand side, along with the other time derivative terms, ${}^{B}\dot{v}_{BO}$ and ${}^{B}\dot{\omega}_{B}$. Let:

$$ {}^{B}_{W}T = \begin{bmatrix} {}^{B}_{W}R & 0 \\ 0 & {}^{B}_{W}R \end{bmatrix} \text{ and } M_I^{-1} = \begin{bmatrix} \frac{I_3}{m} & 0 \\ 0 & {}^{B}I_T^{-1} \end{bmatrix}, \tag{2.55} $$

then the complete non-linear equations of motion for any aircraft can be expressed in state space form as follows [Ref. 11]:

$$ \frac{d}{dt}\begin{bmatrix} {}^{B}v_{BO} \\ {}^{B}\omega_{B} \end{bmatrix} = \chi^{-1}\left\{ \left[\left[\begin{bmatrix} -{}^{B}\omega_{B}\times & 0 \\ 0 & -{}^{B}I_T^{-1}({}^{B}\omega_{B}\times({}^{B}I_T{}^{B}\omega_{B} + I_P{}^{B}\omega_{P})) \end{bmatrix} + \right.\right. \right. $$

$$ \left. M_I^{-1}{}^{B}_{W}T\bar{q}\bar{S}\frac{\partial C_F}{\partial x'}M' \right] \begin{bmatrix} {}^{B}v_{BO} \\ {}^{B}\omega_{B} \end{bmatrix} + M_I^{-1}\left\{ \begin{bmatrix} {}^{B}F_{GRAV} \\ 0 \end{bmatrix} + \right. $$

$$ \left.\left. \begin{bmatrix} {}^{B}F_{PROP} \\ {}^{B}N_{PROP} \end{bmatrix}\delta_T + {}^{B}_{W}T\bar{q}\bar{S}(C_{F_0} + \frac{\partial C_F}{\partial \Delta}\Delta) \right\} \right], \tag{2.56} $$

$$ {}^{U}\dot{P}_{BO} = {}^{U}_{B}R{}^{B}v_{BO}, \tag{2.57} $$

and:

$$ \dot{\Lambda} = S(\Lambda){}^{B}\omega_{B}, \tag{2.58} $$

where:

$$ \Lambda = \begin{bmatrix} \Phi \\ \Theta \\ \Psi \end{bmatrix} \tag{2.59} $$

$$ S(\Lambda) = \begin{bmatrix} 1 & -\cos\Phi\tan\Psi & \sin\Phi\tan\Psi \\ 0 & \cos\Phi\sec\Psi & -\sin\Phi\sec\Psi \\ 0 & \sin\Phi & \cos\Phi \end{bmatrix} \tag{2.60} $$

and:

$$ \chi = I_6 - M_I^{-1}{}^{B}_{W}T\bar{q}\bar{S}\frac{\partial C_F}{\partial \dot{x}'}\dot{M}'. \tag{2.61} $$

P is the position vector of the aircraft, and $S(\Lambda)$ is the matrix of kinematic differential equations based on Euler angles.

III. COMPUTER MODELING

Now that the full non-linear equations of motion have been developed, the next step is to model the aircraft on a computer. Notice that Equations 2.56, 2.57, and 2.58 are in a generic format. That is, they could be used to represent the AROD or the Bluebird or any propeller driven aircraft provided the correct values are used for the stability and control derivatives, $\frac{\partial C}{\partial x'}$, $\frac{\partial C}{\partial \dot{x}'}$, and $\frac{\partial C}{\partial \Delta}$, as well as the forces and moments due to thrust, ${}^{B}F_{PROP}$ and ${}^{B}N_{PROP}$. For this reason, it is convenient to create a model which accepts these values from a generic input file. This allows the same model to be used for different aircraft by simply changing the input data to correspond to the new aircraft. Validation of the model can then be accomplished by entering the appropriate data for a well known aircraft, such as a Cessna, and comparing the results of the model to existing data.

For this report it was desirable to begin with an existing computer model that had already been tested in hardware-in-the-loop simulation so that the results could be compared. The model and controller chosen for the AROD were developed and tested by N. Sivashankar. The SystemBuildmodel he developed is explained here since he chose not to present it in his report [Ref. 3]. For his hardware-in-the-loop test, he developed C code for the controller to run on a 386 PC and developed a model of AROD in VisSim to run on a 486 PC. His hardware-in-the-loop setup is outlined later in Chapter VI Section A and in his report.

The SIMULINK model developed in this section was not used to develop a controller and is presented here as an example of how to implement the equations of motion in a SIMULINK block diagram. For an example of how to implement these

equations in SIMULINK for the Bluebird see [Ref. 1].

A. BASIC NONLINEAR MODEL

The basic nonlinear model is essentially the same for both SIMULINK and SystemBuild. The state derivative equations, Equation 2.56, are implemented and fed into an integrator block which feeds back into the derivative equations.

1. Basic SIMULINK Model

The SIMULINK model is shown in Figure 3.1, and is simply a block representing the state derivative equations, Equation 2.56, and an integrator block in a feedback loop. The SIMULINK implementation of the equations of motion is simplified by using a MATLAB function block. The program listing for this function block is given in Appendix A. Notice that the stability and control derivatives as well as the forces and moments due to thrust are found in a separate MATLAB script file. Appendix A shows this file with the values for AROD in a hover. This MATLAB function has deliberately not been optimized to clearly show how the equations of motion are implemented. The forces and moments due to thrust were measured by B. Stoney, [Ref. 12], and are given by:

$$T_{PROP} = 0.0297\ \delta_{rpm} - 104.7, \tag{3.1}$$

and:

$$l_{PROP} = -0.0542\ T_{PROP} - 0.9138, \tag{3.2}$$

2. Basic SystemBuild Model

The state derivative equations could be implemented on SystemBuild in a similar manner using an user code block. This involves writing a C or FORTRAN

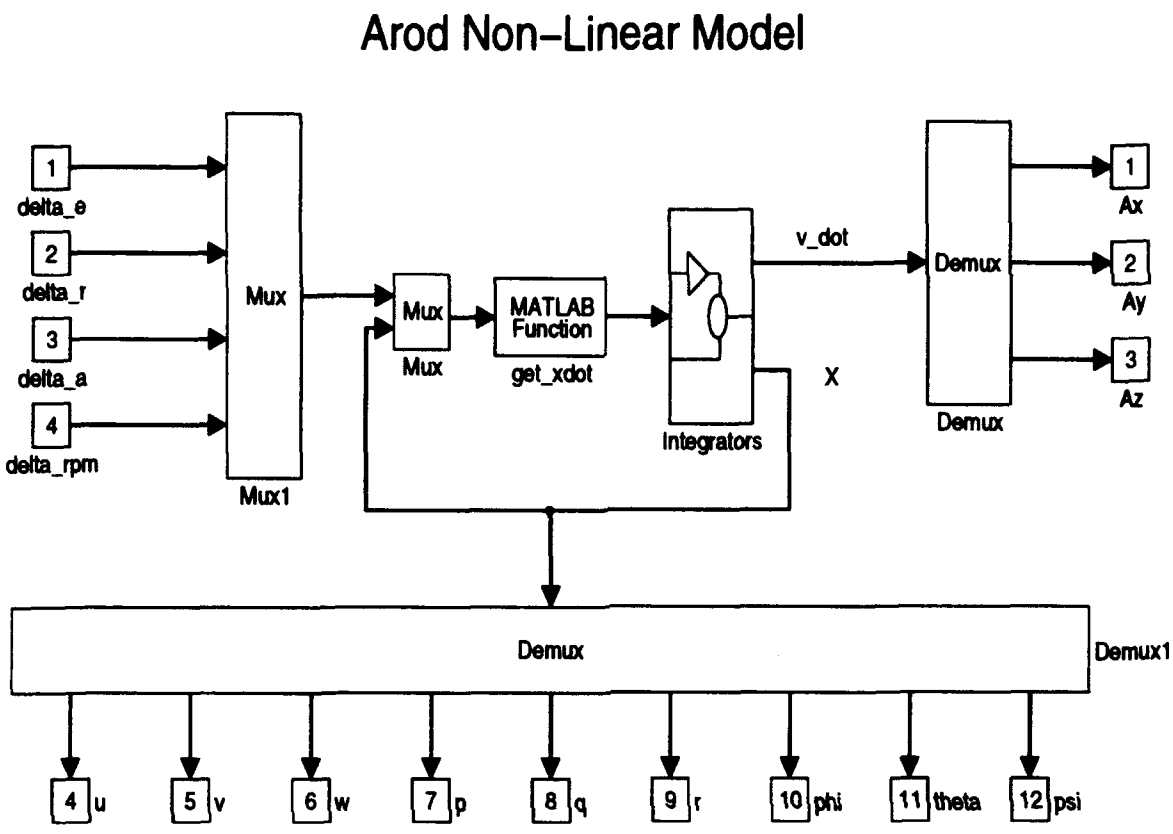


Figure 3.1: SIMULINK Block Diagram of the Equations Of Motion

function which is then linked into the SystemBuild diagram. The Bluebird model used for hardware-in-the-loop testing was developed in this way by J. Byerly. For a detailed explanation of how to implement the nonlinear model with user code blocks see his report [Ref. 13].

The nonlinear SystemBuild model of AROD is shown in Appendix B. The highest level consists of an input block for the constants, a SuperBlock for the aircraft kinematics, and a SuperBlock for all of the integrators. The kinematics SuperBlock is made up of three SuperBlocks representing the angular velocity equations, the linear velocity equations, and the Euler angular rates. The 'L_dot_eq' SuperBlock implements Equation 2.58 directly. The 'lin_velocity_eq' SuperBlock adds Equation 2.31,

Equation 2.51, and the first term of Equation 2.53. The result is the linear portion (the $\frac{d}{dt}{}^B v_{BO}$ portion) of Equation 2.56. The 'ang_velocity_eq' SuperBlock implements the angular portion (the $\frac{d}{dt}{}^B \omega_B$ portion) of Equation 2.56. These values are then fed into an integrator block to determine the states.

B. DISCRETE MODEL

Since the AC100 Model C30 can not automatically generate code for a continuous system, the model must be discretized. The goal is to simulate a continuous time system using a discrete time system. SIMULINK and SystemBuild do this by using a very small time step size with a continuous type integration algorithm. The continuous model can be discretized in SystemBuild with the 'Transform SuperBlock' option under the 'Build' menu. Simply choose a small step time and the SuperBlocks are automatically transformed. The only difference is that all integrators will be replaced by 'discrete' integrators as shown in Figure 3.2, where T is the step time chosen for the discrete system.

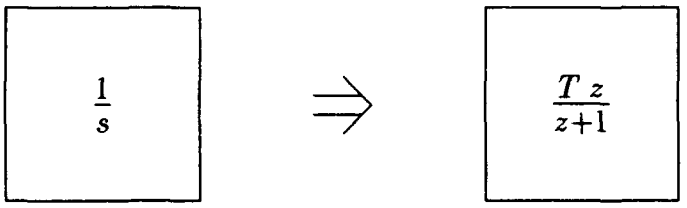


Figure 3.2: Transformation of an Integrator from Continuous Time to Discrete Time

The step time must be evaluated with the model to determine the optimum step time. If the step time is too large the discrete system will not be an accurate

model. If the step time is too small the computation time necessary may slow down the discrete system to the point where it becomes unstable.

C. TESTING THE MODEL

The performance of the aircraft model must be verified. This is usually accomplished by replacing the stability and control values with those of a well known aircraft such as a Cessna. The nonlinear model can then be trimmed at a given flight condition and the eigen values of the resulting linear model can be compared to existing data.

1. Testing the SIMULINK Model

The SIMULINK model of AROD presented here was trimmed for the hover condition and linearized. The resulting state-space matrices were identical to the state-space matrices determined by D. Kuechenmeister [Ref. 2]. Since this model was not used to develop a controller, no further testing was completed.

2. Testing the SystemBuild Model

The SystemBuild model of AROD developed by N. Sivashankar also produced the same state-space matrices when trimmed and linearized for hovered flight. Refer to his report for more details on the testing of his model [Ref. 3].

IV. DESIGN AND SOFTWARE TESTING OF THE CONTROLLER

Now that a valid linear model has been developed, the controller can be designed. First, the designer determines which states or outputs are available for feedback and the control inputs to be used by the controller. For the AROD, the following states are measured:

$$x = \begin{bmatrix} p \\ q \\ r \\ \theta \\ \psi \end{bmatrix} \tag{4.1}$$

The inputs are elevator, rudder, aileron, and rpm (revolutions per minute of the propeller):

$$\Delta_{Input} = \begin{bmatrix} \delta_e \\ \delta_r \\ \delta_a \\ \delta_{rpm} \end{bmatrix} \tag{4.2}$$

H_∞ synthesis was used to design the state feedback controller. It is outlined in the next section.

A. H_∞SYNTHESIS MODEL

Consider Figure 4.1. Here w represents exogeneous inputs, z represents regulated outputs, P represents the plant model, y represents the actual plant output, and u_c is the control input created by the controller. Using the notation in [Ref. 14], suppose the plant is partitioned as:

$$P = \begin{bmatrix} P_{11} & P_{12} \\ P_{21} & P_{22} \end{bmatrix}, \tag{4.3}$$

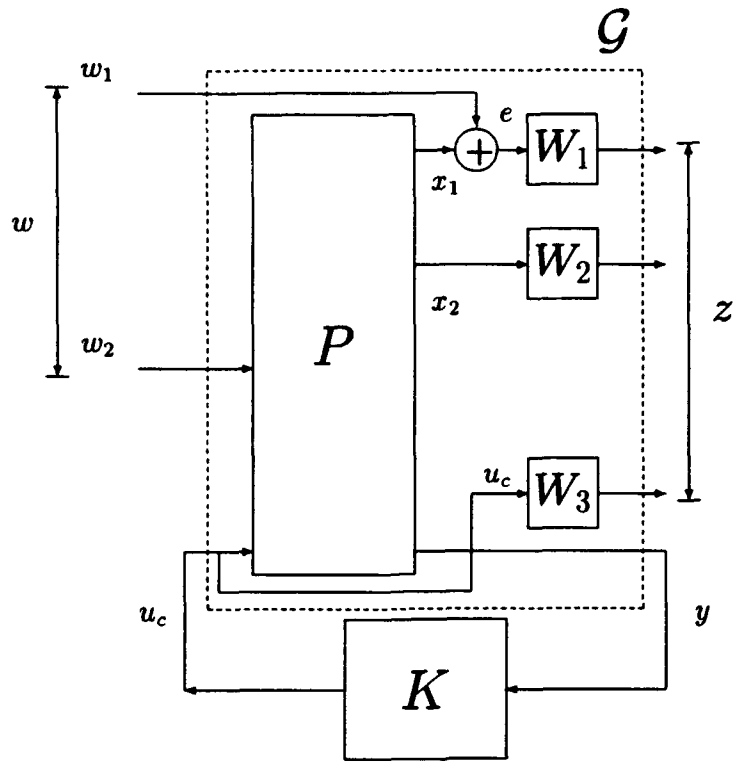


Figure 4.1: H_∞ Synthesis Model

so that:

$$z = P_{11}\, w + P_{12}\, u \qquad y = P_{21}\, w + P_{22}\, u \tag{4.4}$$

Then u and y can be eliminated using $u = Ky$, to obtain:

$$z = \left[P_{11} + P_{12}\, K\, (I - P_{22}\, K)^{-1}\, P_{21}\right]\, w, \tag{4.5}$$

This is normally denoted by:

$$z = F_l(P, K)\, w \tag{4.6}$$

The H_∞ optimization problem is then:

$$\text{Find } K \text{ which minimizes } \|F_l(P, K)\|_\infty \tag{4.7}$$

For the AROD, the input w_1 shown in Figure 4.1 is defined as:

$$w_1 = \begin{bmatrix} \text{roll rate command} \\ \text{pitch command} \\ \text{yaw command} \end{bmatrix} \tag{4.8}$$

and the input w_2 is:

$$w_2 = \text{rpm input} \tag{4.9}$$

The control commands are:

$$u_c = \begin{bmatrix} \text{elevator} \\ \text{rudder} \\ \text{aileron} \end{bmatrix} \tag{4.10}$$

The signals x_1 and x_2 are:

$$x_1 = \begin{bmatrix} p \\ \Theta \\ \Psi \end{bmatrix} \qquad x_2 = \begin{bmatrix} q \\ r \end{bmatrix} \tag{4.11}$$

The designer changes the cost function weights W_1, W_2, and W_3 to obtain the desired bandwidth in the command and control loops. [Ref. 15]

B. DISCRETE CONTROLLER

The controller obtained using H_∞ synthesis has the following state-space representation:

$$C : \{ \begin{array}{rcl} \dot{x}_c & = & B_c(y_1 - y_c) \\ u & = & C_c\, x_c + D_c\, y_2 \end{array} . \tag{4.12}$$

Since C will be implemented on the digital computer it must be discretized first:

$$C_D : \{ \begin{array}{rcl} (x_c)_{K+1} & = & \Delta T\, B_c\, (y_1 - y_c)_K \\ u_K & = & C_c\, x_{c_K} + D_c\, y_{2_K} \end{array} , \tag{4.13}$$

where ΔT is the sampling period of the discrete time system.

1. SystemBuild Discrete Controller

The SystemBuild implementation of a discrete controller uses a state-space block with the appropriate values as a matrix gain. The discrete controller for AROD

is shown in Appendix B Figure B.4. Notice that for this implementation ΔT has been multiplied into the B matrix instead of the control gain matrix.

C. CLOSED–LOOP SOFTWARE TESTING

The procedure for closed-loop testing of the controller is basically the same for both continuous and discrete time systems. The model and controller are connected as shown in Figure 4.2. Note that the discrete time controller could be tested with the continuous time model if the outputs of the discrete controller are routed through a zero-order hold before being input to the continuous model. This step is done automatically by SystemBuild when discrete and continuous SuperBlocks are connected within the same block diagram.

1. SystemBuild Testing

SystemBuild testing can be accomplished in several ways. All of these methods require the user to define a time vector. For a 40 Hertz controller the time vector might be:

$$t = 0 : 0.025 : 20; \tag{4.14}$$

Which produces a vector of 801 elements, starting at zero, spaced at 0.025 seconds, and ending after 20 seconds. The user can define an input vector in MATRIX_X or connect signal generator blocks inside the model. The user then selects 'Analyze SuperBlock' from the 'Build' menu and enters the appropriate values. Typing 'sim' at the MATRIX_X prompt will begin the test and create a matrix of output values. The output matrix can be broken into vectors and observed using the 'plot' command. The results of this test for the AROD are presented in [Ref. 3].

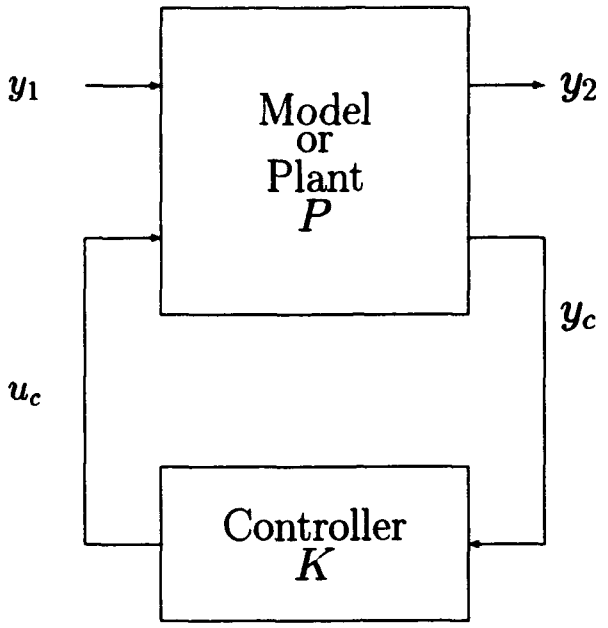


Figure 4.2: Closed-Loop Block Diagram

2. AC100 Model C30 Testing

The discrete model can be tested on the AC100 Model C30 by following the procedures outlined later in Chapter VI Section B without connecting any of the hardware. The closed-loop connections are left in the model and the desired outputs are selected for the Interactive Animation display. The test results will be identical to those found above since the C30 processor is using the exact same closed-loop system as SystemBuild.

V. MODELING ACTUATORS AND SENSORS

For both the AROD and the Bluebird, all of the control surfaces are actuated by Futaba FP–S34 servo motors. These actuators where originally modeled by Sandia Labs as a second order system with $\zeta = 0.6$ and $\omega_n = 20$ radians.

$$H(s) = \frac{\omega_n^2}{s^2 + 2\,\zeta\,\omega_n\,s + \omega_n^2}, \tag{5.1}$$

This section examines the development of an accurate model for these actuators.

A. ACTUATOR STEP RESPONSE

The step response of a system can be used to determine its transfer function [Ref. 16]. An example step response is shown in Figure 5.1. This response is typical for an underdamped second order system. To determine the transfer function, it is necessary to determine the values for M_p and t_r.

The measured step response for the Futaba servos is shown in Figure 5.2 with the step response for the transfer function given in Equation 5.1. This step response was measured using the data acquisition feature of the AC100 Model C30 and the test setup outlined in the next chapter.

The values for M_p and t_r were measured as:

$$M_p = 0.1197 \qquad t_r = .059\ , \tag{5.2}$$

With these values a second order transfer function can be created by calculating the natural frequency ω_n and the damping ratio ζ using the following formulae:

$$\zeta = \frac{ln(M_p)}{\sqrt{ln(M_p)^2 + \pi^2}}, \tag{5.3}$$

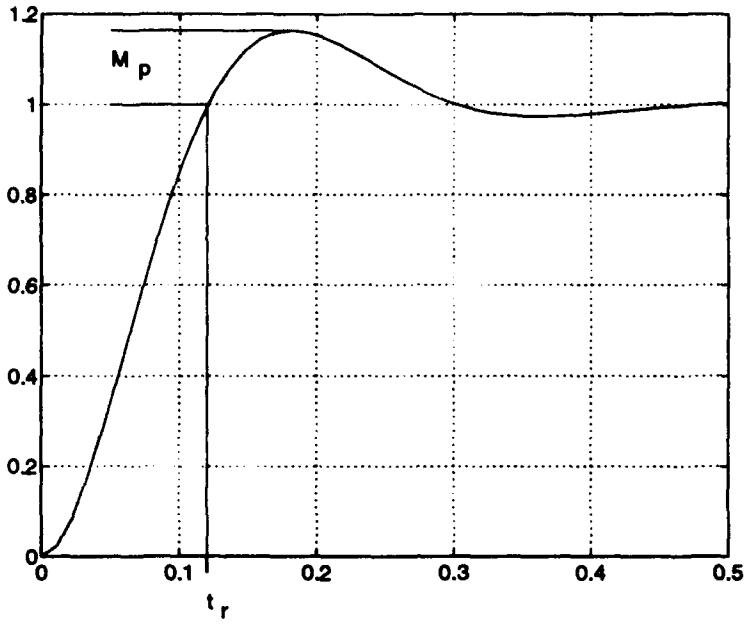


Figure 5.1: Typical Step Response of an Underdamped System

and:

$$\omega_n = \frac{-tan^{-1}(\frac{\sqrt{1-\zeta^2}}{\zeta})}{\sqrt{1-\zeta^2}\, t_r}, \tag{5.4}$$

These calculations resulted in:

$$\omega_n = 19.94 \qquad \zeta = .559 \ , \tag{5.5}$$

Since the step response did not match well with the step response of the calculated transfer function, a limited frequency response of the actuators was measured.

B. ACTUATOR FREQUENCY RESPONSE

The actuators were given a sinusoidal command input and the response was measured for frequencies of 1, 2, 3, and 4 Hertz. This procedure could have been duplicated for many more frequencies and the result would be a complete frequency response for the actuators. Figure 5.3 shows the frequency response at 3 Hertz. Each

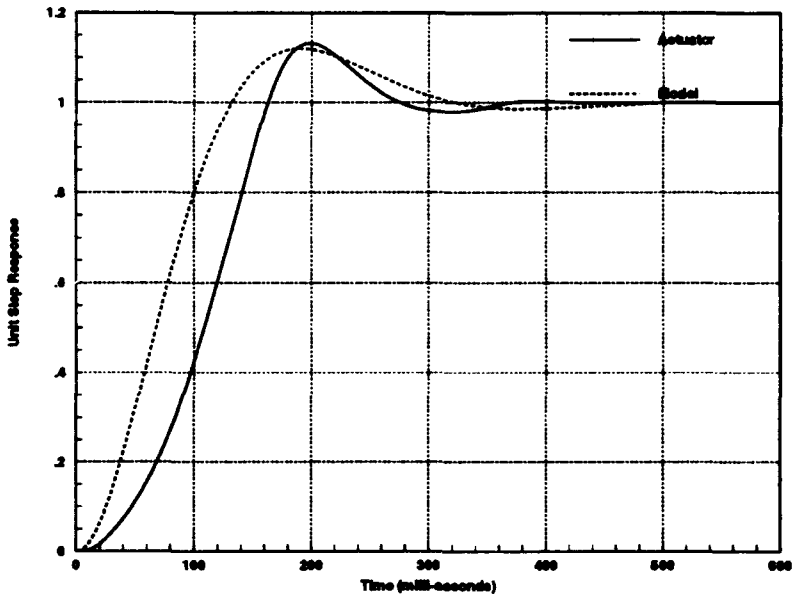


Figure 5.2: Step Response of Actuator and Second Order Model

pole in a system will result in a total of 90 degrees of phase shift in the frequency response and half, or 45 degrees, of this shift occurs at the natural frequency [Ref. 16]. Since the measured phase difference was approximately 180 degrees at 3 Hertz, the servo motors are more accurately modeled as fourth order systems.

One possible fourth order model was determined and is given in Figure 5.4. Figure 5.5 shows the step response of the actuator and the fourth order model step response.

C. ACTUATOR SENSORS

Since the hardware-in-the-loop test will not cause the aircraft to move, aircraft sensors such as Inertial Measuring Units and Air Data Sensors can not be used.

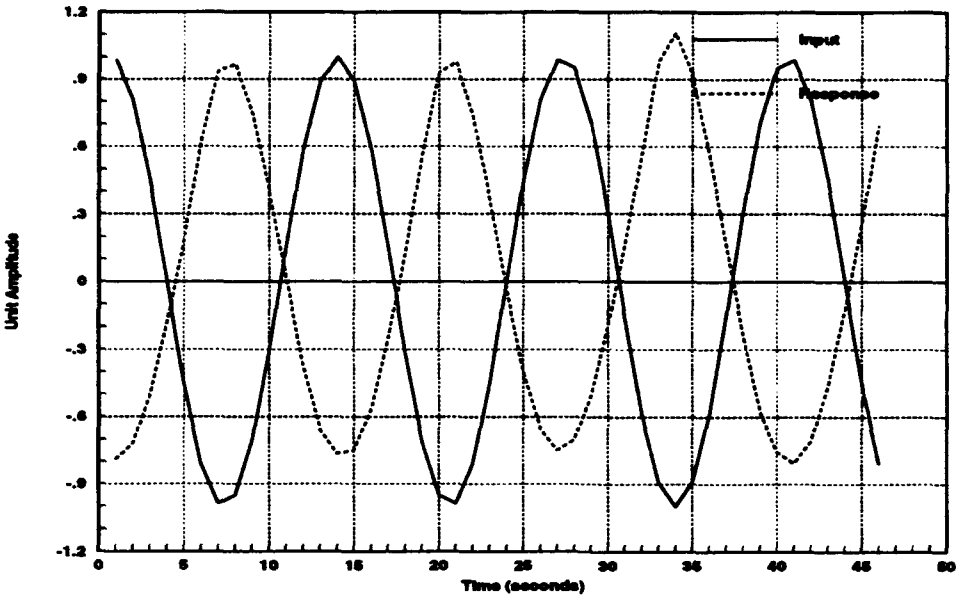


Figure 5.3: Frequency Response of an Actuator at 3 Hertz

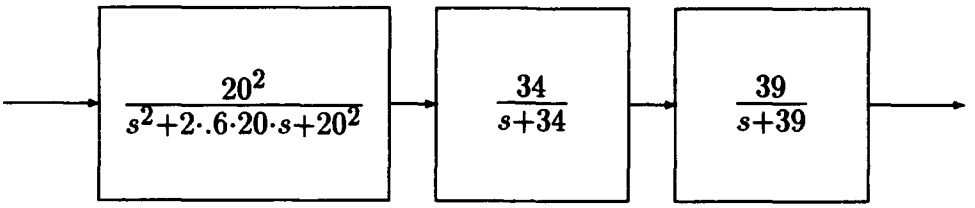


Figure 5.4: Fourth Order Actuator Model

Therefore the actuator positions must be determined. This is accomplished using an angular position sensor. The measured vane positions can then be used to determine the states of the aircraft so that an hardware-in-the-loop test can be preformed.

The Futaba servos used do not include a separate position sensor. There is an internal control circuit which determines which direction and how far to move the servo based on the input signal. This input signal is a Pulse Width Modulated, or

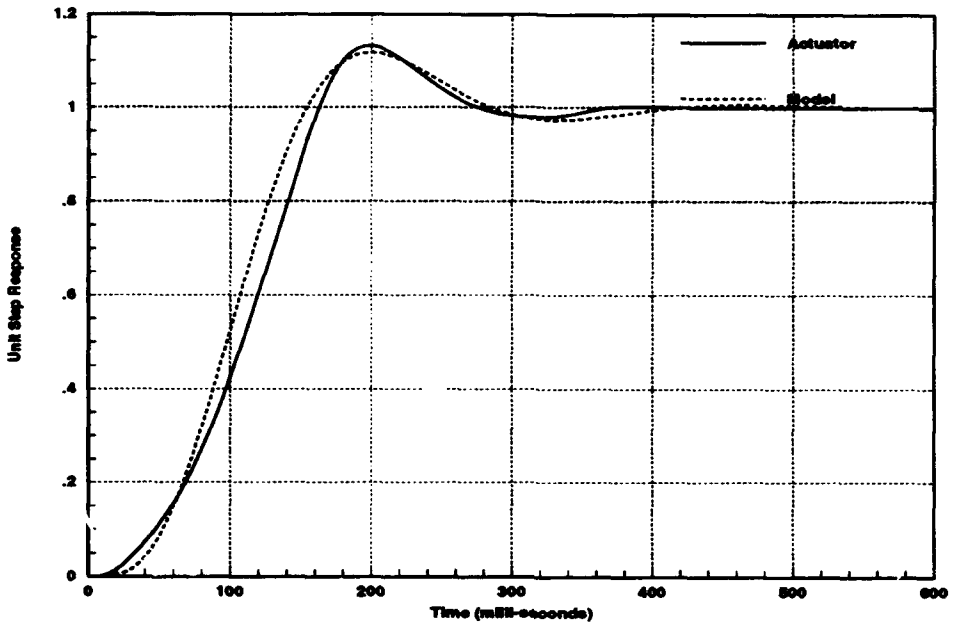


Figure 5.5: Step Response of Actuator and Fourth Order Model

PWM, pulse. The length of the pulse determines how far the servo is to turn. The previous hardware-in-the-loop test defined the throw of these actuators as being from 0 to 200 degrees with the center position at 100 degrees. For this report the center position is defined as 0 degrees with full throw being plus or minus 100 degrees. A pulse width of approximately 0.3 milli-seconds corresponds to -100 degrees while a pulse width of approximately 2.4 milli-seconds corresponds to $+100$ degrees. The internal control circuit includes a small potentiometer in a feedback loop to control the motor. The servos were modified to include wires connected to the center and ground leads of this potentiometer as shown in Figure 5.6. The voltage across these two wires was then measured and divided by 5 volts, (the supply voltage), to determine the position as a ratio of the total allowable motion.

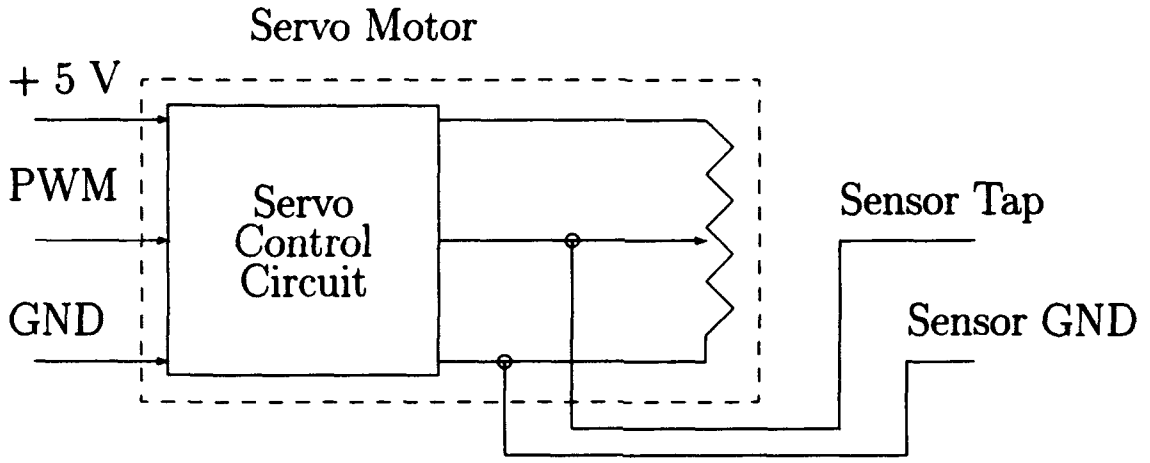


Figure 5.6: Actuator Sensors

This sensor design depends on a constant 5 volts being applied to the positive end of the potentiometer. Since the same voltage also supplies power to the servo motor, this voltage actually changes slightly while the motor is turning. The servo motors draw approximately 8 milli-amperes of current when they are not moving and up to 200 milli-amperes when they are moving. The increased current draw during motion causes a drop in the supply voltage. Since the measured voltage is always divided by 5 volts the result is a noisy sensor position. This noise was measured as approximately one half of one degree in each of the four vanes. Since these sensed positions are used to determine the aircraft states, noise enters all of the states. The most pronounced effect of this noise shows up in the roll-rate p because all of the vanes add together to determine the aileron command. A 0.5 deg change in all of the vanes from one measurement to the next is equivalent to an aileron control surface movement of 80 degrees per second.

To reduce this noise an additional wire was added so that the positive voltage

on the potentiometer could be measured at the same time as the center voltage. Figure 5.7 shows the new sensor wiring.

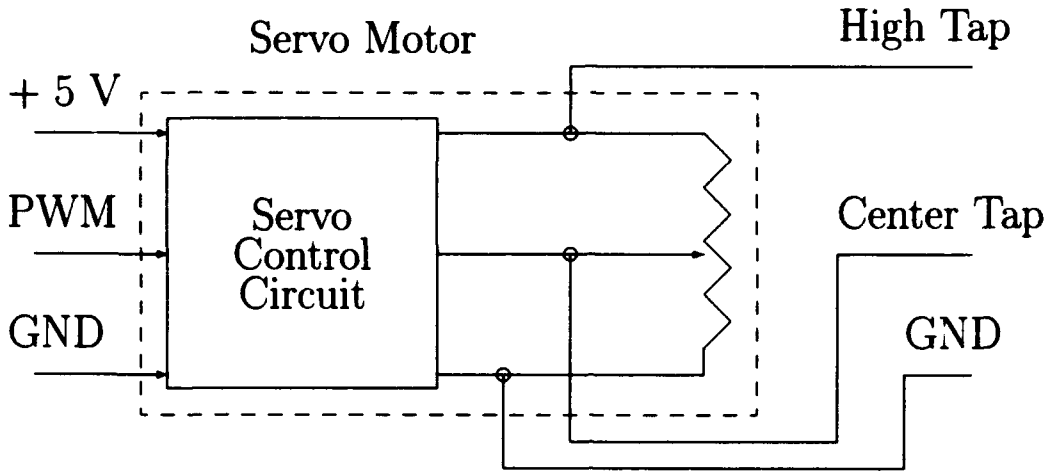


Figure 5.7: Modified Actuator Sensors

Now the ratio of the voltage measured from the ground to the center tap, over the voltage from the ground to the high tap gives the position as a percentage of total motion.

$$\frac{V_{CenterTap}}{V_{HighTap}} = \% \text{ of total motion} \tag{5.6}$$

The result of the new sensor design was an order of magnitude reduction in the sensor noise. Figure 5.8 shows the measured response of an actuator to a 2 Hertz sine wave input. The two wire response was measured prior to adding the additional wire to the sensor. The responses have been time shifted for clarity.

D. UNDER–SAMPLING

When continuous time signals are sampled at less than the Nyquist frequency and then reconstructed, the resulting waveform will have low frequency components.

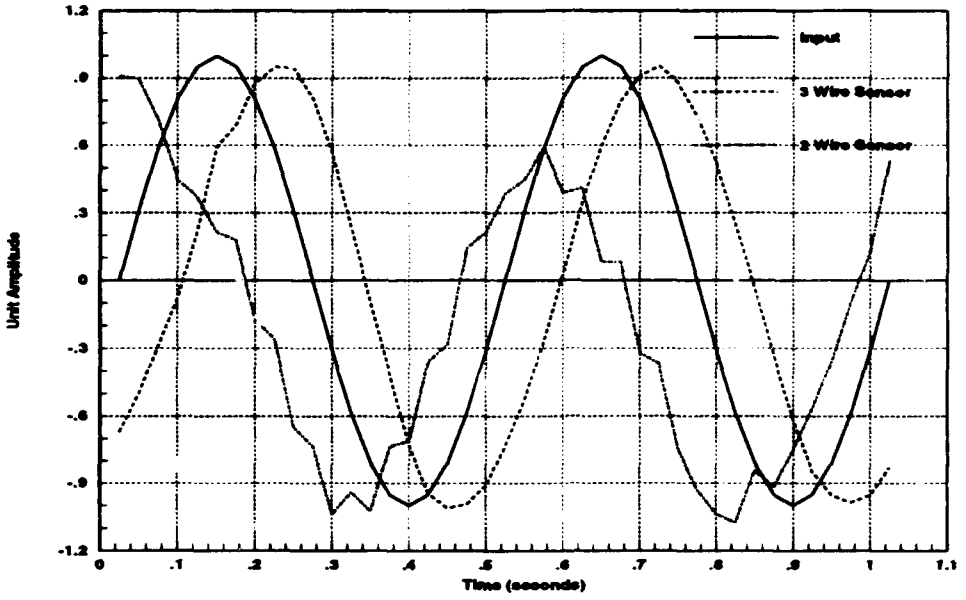


Figure 5.8: Noise Comparison of Actuator Sensors

This effect is known as under-sampling [Ref. 17].

A common example of under-sampling can be seen on television. The video camera essentially takes samples of the continuous time world and presents them in a discrete fashion. The human eye captures these discrete pictures and filters them so that the mind perceives continuous motion. When the video camera samples a rotating object such as a wagon wheel at a frequency less than the Nyquist rate for the rotation speed, the wheel may appear to turn backwards. Figure 5.9 shows how a 37 Hertz sine wave, sampled at 20 Hertz, appears to be a 3 Hertz sine wave. The solid line is the continuous time sine wave, the asterisk symbols are the 20 Hertz samples, and the dashed line is the continuous time estimate of the samples. Notice that the reconstructed wave starts out negative. If this were a mathematical representation of

the wagon wheel, it would appear to turn backwards. The effects of under-sampling were eliminated in the AROD by the technique used to reduce aliasing which is discussed next.

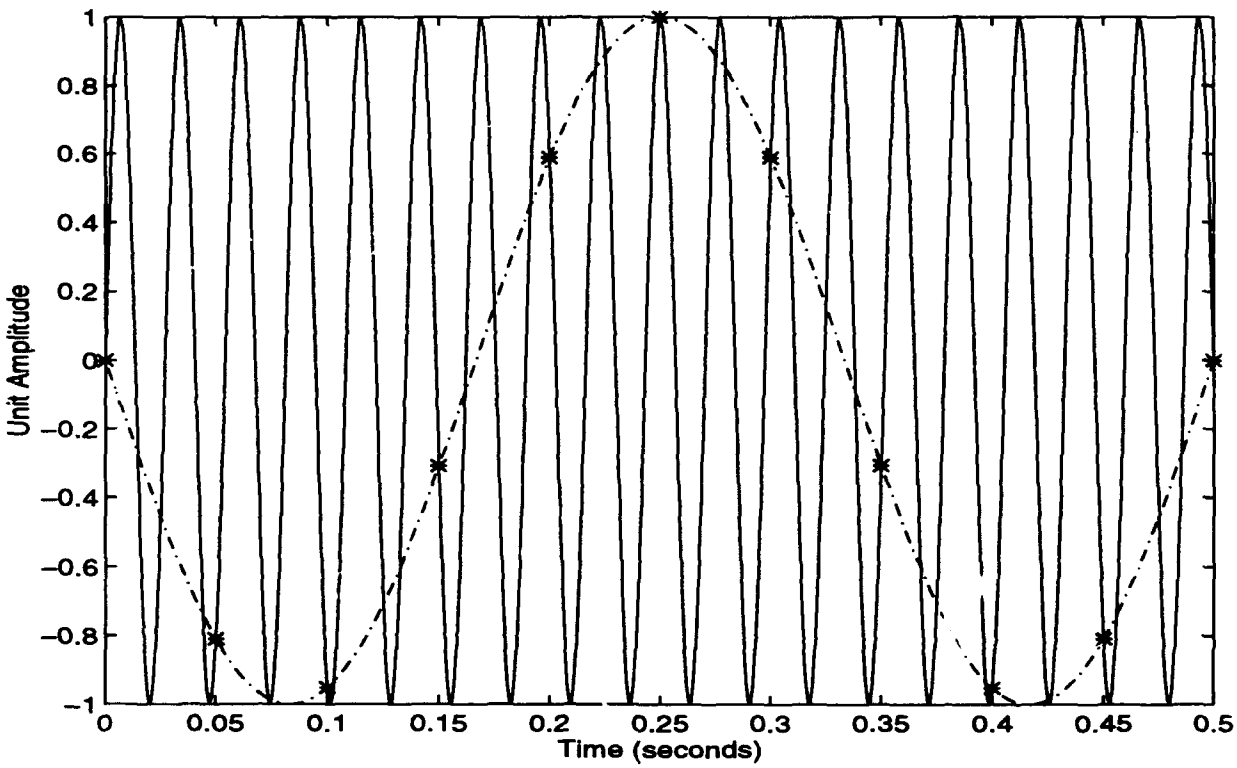


Figure 5.9: Under–Sampling of a Continuous Time Signal

E. ANTI–ALIASING

Sampling a continuous time signal results in an infinite train of 'copies' of the sampled signal repeating at integer multiples of the sampling frequency [Ref. 17]. Aliasing occurs when the sampling frequency is such that these 'copies' overlap. Figure 5.10 shows the frequency response of an example signal, the sampled frequency response when sampled at 50 Hertz, and the sampled frequency response when sampled at 16 Hertz. The area between the triangles in the third response is called

aliasing.

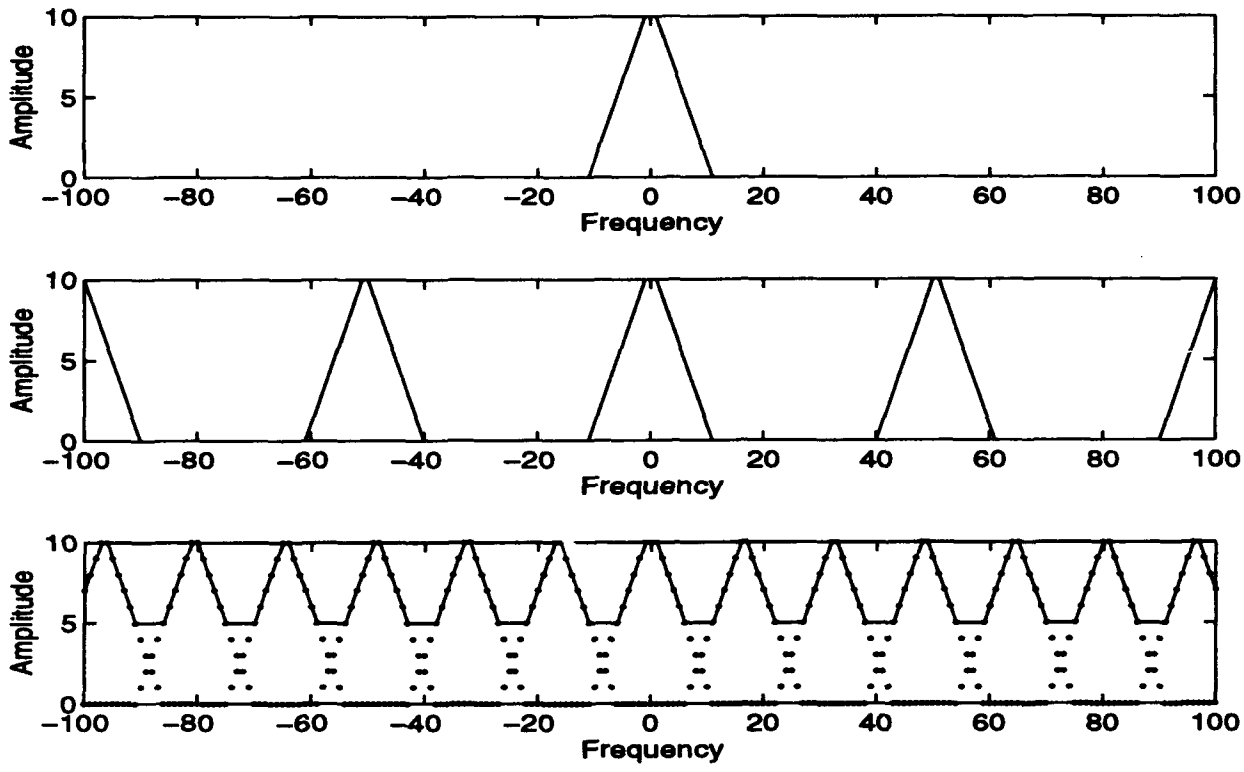


Figure 5.10: Example of a Continuous Time Signal and Aliasing

The effects of aliasing were reduced in the AROD hardware-in-the-loop test by adding an anti-aliasing filter. The sensor voltages were sampled at 1000 Hertz. The ratio of the two voltages was then fed into a third order low-pass Butterworth filter with a cut-off frequency of 10 Hertz. The output of this filter is then sampled at 40 Hertz eliminating aliasing of any noise in the frequency range of interest. Appendix B includes the SystemBuild block diagrams for the anti-aliasing filters. The discrete low-pass filter was implemented using the 'FIIR' command in MATRIX$_X$. The resulting state-space matrix was then put into the SystemBuild block diagram. A separate filter was necessary for each of the four vanes. The state-space representation of the

filter is given by:

$$F = \left[\begin{array}{c|c} A_F & B_F \\ \hline C_F & D_F \end{array}\right] \tag{5.7}$$

The values for the discrete Butterworth low-pass filter state-space matrix were:

$$F = \left[\begin{array}{ccc|c} 1.9353D+00 & -9.3912D-01 & 0.0000D+00 & 2.9146D-05 \\ 1.0000D+00 & 0.0000D+00 & 0.0000D+00 & 0.0000D+00 \\ 3.9353D+00 & 6.0879D-02 & 9.3906D-01 & 2.9146D-05 \\ \hline 3.9353D+00 & 6.0879D-02 & 1.9391D+00 & 2.9146D-05 \end{array}\right], \tag{5.8}$$

For the AROD, the frequency response of interest is from 0 to 3 Hertz (approximately 20 radians). The noise is estimated to have major components at 40 Hertz and harmonics of 40 since the PWM frequency is 40 Hertz. By sampling at 1000 Hertz, we are assuming that the noise level is insignificant for frequencies above 500 Hertz. The 10 Hertz cutoff frequency on the filter is designed to eliminate the noise at, and above 40 Hertz.

VI. HARDWARE–IN–THE–LOOP TESTING

The most critical phase of testing is the hardware-in-the-loop test of a controller. This is usually the final validation of a controller prior to an actual flight test. Typically this involves the actual control computer, actuators and some or all of the actual sensors. In the case of the AROD, the only sensors that can be used for this test are the servo-motor position sensors. Since no motion is involved, the IMU, GPS, and Air Data sensors would not produce usable data and therefore these sensors must be modeled along with the aircraft.

The controller is typically implemented on a microprocessor capable of interfacing with the required hardware. In this case a 486 PC type computer is the intended control computer. For the first hardware-in-the-loop test, the AC100 Model C30 will serve as the control computer and as the plant model computer. Later, the controller will be separated and implemented on the 486 PC. Before discussing the new hardware-in-the-loop test setup, the previous test setup is presented for comparison.

A. PREVIOUS TEST SETUP

Before automation of hardware-in-the-loop testing, the aerospace controls engineer had to rely on computer scientists or know how to program in a high level language. For his hardware-in-the-loop test, N. Sivashankar wrote C code for the controller and for all of the necessary I/O drivers. His setup is presented in his report, [Ref. 3], and briefly outlined here. The complete setup is shown in Figure 6.1. The 386 PC runs the controller and outputs PWM signals to the vanes. The 486 PC

senses the vane positions and computes the new states. The 486 PC then sends out analog signals to simulate the angular rate sensors and the euler angle sensors.

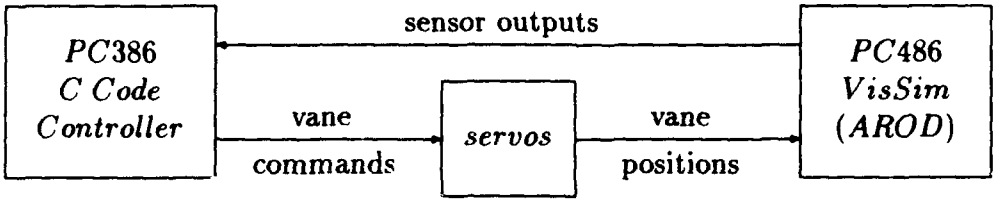


Figure 6.1: Previous Setup for Hardware–In–The–Loop Simulation

The basic configuration of the 386 PC is shown in Figure 6.2. The 386 PC reads the analog inputs and converts the measured values to the correct units for the controller. The new control commands are then computed and sent out by the counter/timer board as PWM signals.

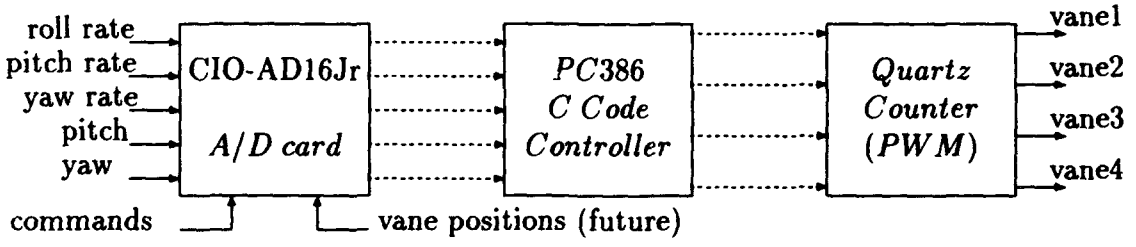


Figure 6.2: Configuration of the Data Acquisition Cards on the 386 PC

The configuration of the 486 PC is shown in Figure 6.3. The vane sensor voltages are read by VisSim and then used to calculate the new aircraft states. The angular rates p, q, and r and the angles θ and ψ are then sent out as analog signals to the 386 PC simulating the angular sensors.

The most significant problem of this hardware-in-the-loop test setup was the speed of the AROD model. The VisSim model of AROD could not be run in real time,

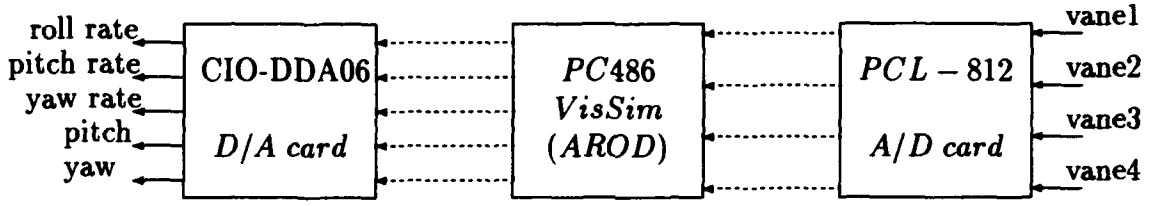


Figure 6.3: Configuration of the Data Acquisition Cards on the 486 PC

thus the controller had to be artificially slowed to 4 Hertz. The control algorithm was implemented in C-code as a function call which was driven by an interrupt. By design, this interrupt should have been at a rate of 40 Hertz. Since the VisSim model could not produce updated states at this rate, the interrupt was slowed to 4 Hertz. In this way, the controller performed as if it were running at 40 Hertz while actually running at 4 Hertz. For more details refer to [Ref. 3].

B. AC100 GRAPHICAL USER INTERFACE

The new hardware-in-the-loop test setup utilizes Integrated Systems, Incorporated's AC100 Model C30. The AC100 Graphical User Interface, or GUI, is the workstation users link to all of the necessary software tools for modeling and testing. Prior to using the GUI, the user must 'source' the 'ac100setup' file. This is done by typing:

'source $ISI/AC100/bin/ac100setup.sh'

at the unix prompt. This line can also be included in the '.login' file so that the 'ac100setup.sh' file is automatically run each time the user logs in to the workstation. This section assumes that the user has manually entered MATRIX$_X$ previously and is using the GUI for the first time now that the controller and model are complete.

The AC100 manuals refer to a model and controller as a project, [Ref. 18, 19, 20] with the project name being the name of the highest level SuperBlock in the diagram. There are standard files which must be present in the local directory for each project which have common names and the AC100 uses file extensions to separate files within a project. It is required to use a separate directory for each project to avoid using the project files from one project with the standard files of another project.

The first of these standard files is the animation configuration file, (animation.cfg). Each project will have a slightly different 'animation.cfg' file, but it must have that name. To create this file, type 'makeproject' at the unix prompt. The program will assume that the project name is the same as the directory name. If this is true, the user may accept all of the default settings by hitting 'enter' at each prompt.

The next standard file is the 'target_config.cfg' file. To create this file the user types 'retarget' at the unix prompt. The user will be asked for the 'ac100hostname' which is either 'AC100' or 'america'. All of the remaining defaults should be selected. These files are only created once for each project. Now that these basic files have been set up, the user types 'ac100' at the unix prompt to start the GUI. The GUI is used with the mouse and a single left mouse button click will activate the selected function. Figure 6.4 shows the AC100 GUI.

The basic project file containing all of the required information about the model and controller including the input and output names is the real-time file. This file is created by selecting 'Generate Real-Time Code' from the SystemBuild 'Build' menu and selecting the top level SuperBlock from the list. The file name can be specified and defaults to the name of the SuperBlock selected with the '.rtf' extension. The user can then exit MATRIX$_X$ and select 'Autocode'.

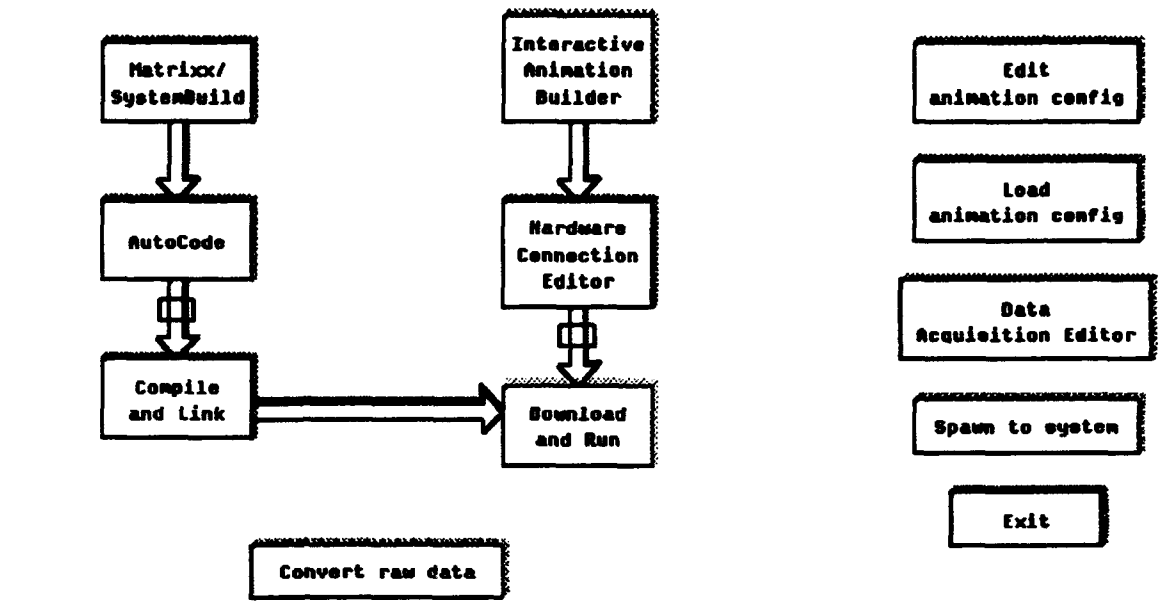


Figure 6.4: AC100 Graphical User Interface

The next step is to build the Interactive Animation, or IA, display to be used during the hardware-in-the-loop test. The user will first need to determine which outputs to display and which inputs are desired for interaction with the running model and controller. The user may need to add inputs and outputs to get the desired results.

1. Interactive Animation

The user clicks on the 'Interactive Animation Builder' block to design the interface screen for working with the C30. The main IA picture for the AROD hardware-in-the-loop test is shown in Figure 6.5. Once in the IA Builder, the user double-clicks on any blank area to bring up the palette of display icons. The Inter-

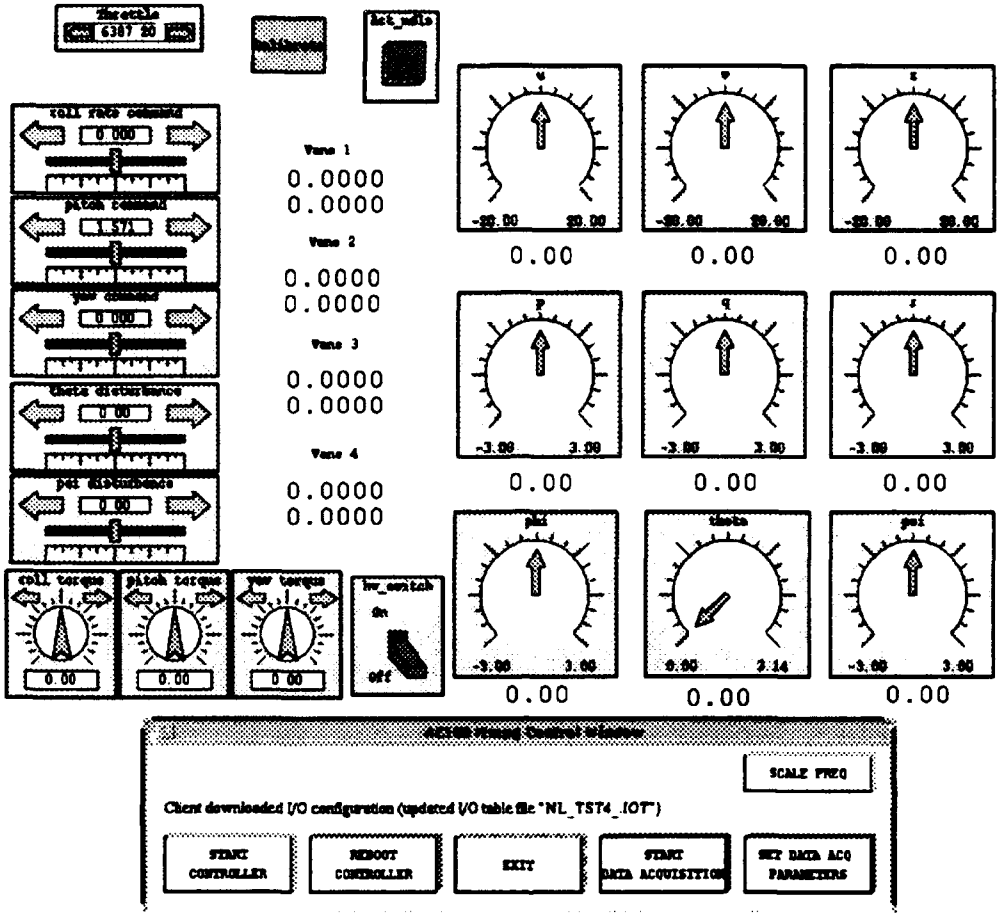


Figure 6.5: Interactive Animation Display for AROD Controller

active Animation section of the SystemBuild manual [Ref. 21] and the AC100 User's Guide [Ref. 19] have details on all of the available icons and how to edit them for specific needs. The user then selects 'Load RTF' and enters the name of the '.rtf' file. The user then connects all of the icons to the respective inputs and outputs using the same connection procedures as in SystemBuild. If the user wants to display an input from one of the hardware connections, i.e., an A/D input, an extra output will have to be added to the model. Inside the model the input is then connected to the new output through an unity gain block. Once the IA picture is complete and all of the inputs and outputs have been connected, the user selects save. Additional

pictures can be created in the same manner. The main picture should be saved as 'project_name.pic'.

Additional pictures, such as a calibration screen, can be 'chained' using the 'process' icon. The user must then edit the 'animation.cfg' file and add 'filename.pic' under the 'PROCESS_PICTURES' section where 'filename' is the name of the 'chained' picture. The IA calibration picture used for the AROD is shown in Figure 6.8.

2. Hardware Connection Editor

The Hardware Connection Editor, or HCE, screen is shown in Figure 6.6 and explained in the AC100 Reference Manual [Ref. 18]. The individual hardware modules are further explained in the AC100 Supplemental Reference Manual [Ref. 22]. The HCE is used to make connections to hardware and also to the IA picture. All connections to the IA picture will be completed automatically and should not be changed. Before invoking the HCE, the user should place a copy of the file 'c_c30.hce' in the project directory. This file can be copied from the 'c:\ac100c30\station' directory on the AC100. The first screen of the HCE deals with inputs to the model and the second screen deals with the outputs. Inputs will initially show 'MONITOR_INPUT' as the 'type' and are changed by selecting 'Device_Type'. If the correct 'c_c30.hce' file is in the current project directory, the user can use the arrow keys or the mouse to select the desired hardware for connection. The module field, 'mod', will change to 1 for all of the hardware options and must be changed to the correct module number. The module numbers differ according to which C30 is being used and are given in Table 6.1. Next the user selects the channel number, 'ch#' which is 1 to 1000 for each of the serial ports and 1 to 16 for both the 'IP_HiADC' and the 'IP_PWM'. Each

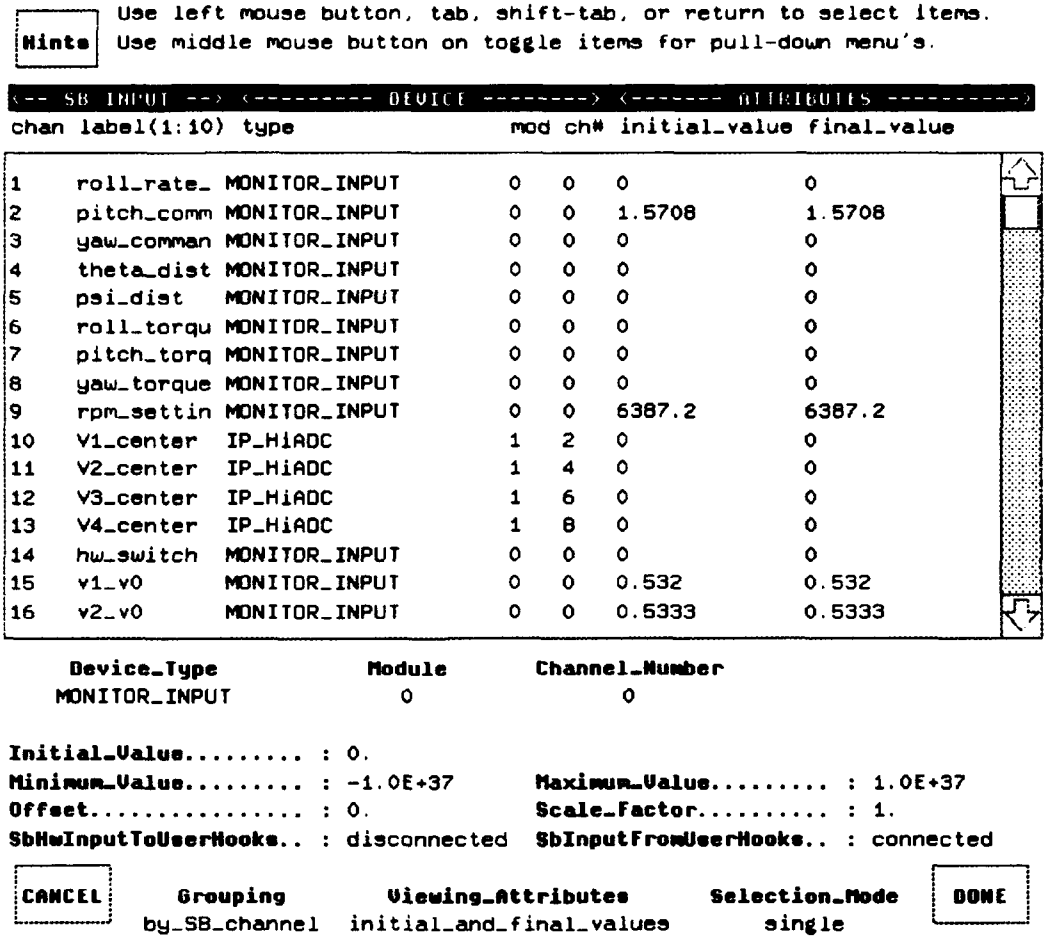


Figure 6.6: Hardware Connection Editor

'IP_SERIAL' module has two serial ports referred to as 'chanA' and 'chanB'. The 'ch#' field refers to the data channel. Each input or output variable will require an individual channel. The AC100 Supplemental Reference Manual [Ref. 22] also talks about the hardware channel number. This number is a fixed value and refers to the hardware address of that I/O device. The outputs are initialized to 'NO_DEVICE' and are connected in a similar manner.

TABLE 6.1: C30 HARDWARE MODULE NUMBERS

	AC100	America
IP_SERIAL	2 or 3	1 or 3
IP_HiADC	1	2
IP_PWM	4	4

a. Serial Connections

The SERIAL modules can be used for input and/or output [Ref. 22] pages 118-135. The serial modules were used for the Bluebird hardware-in-the-loop test and the AROD Flight Management Unit test. The Bluebird has an Inertial Measuring Unit which measures linear accelerations, angular rates, and euler angles. This information is available to the controller through a serial port. The serial information is a 40 byte string of hex characters terminated by a return character. This format differs from the format expected by the C30, however, the user can edit the 'user_ser.c' file and specify any desired format for the data. For more information on serial interfacing with the IMU see [Ref. 23].

b. Analog–to–Digital Connections

The HiADC module is used for input only [Ref. 22] pages 110-117. Any analog voltage signal can be sampled and used by SystemBuild in a digital format. The SystemBuild block diagram can then use the input in units of volts, or convert the number to some other units. Section 3 below discusses the conversion from volts to degrees used for the AROD hardware-in-the-loop test.

c. Pulse Width Modulation Connections

The PWM module has many uses for both input and output [Ref. 22] pages 105-109. Note that the actual name of this module is 'IP_68332' but is referred to here as 'PWM' since this is the only mode used for this report. In the PWM mode, the user specifies the duty cycle as the output from the SystemBuild diagram. The user can also edit the 'c_c30.hce' file to specify the frequency of the pulses. The refresh frequency is integer parameter one which is labeled as I1 (column 10) under the output section of this file. Figure 6.7 depicts the relevant quantities for a PWM signal.

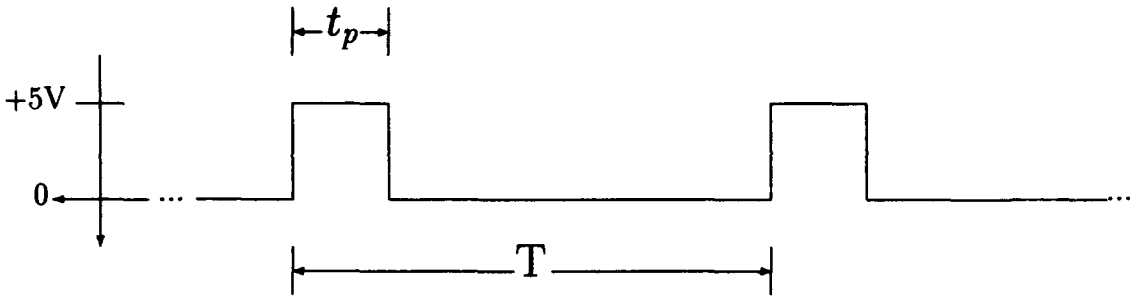


Figure 6.7: Timing Example for Pulse Width Modulation

The spacing from the leading edge of one pulse to the next is called the period T, and is the inverse of the refresh frequency or:

$$T = \frac{1}{f} \tag{6.1}$$

The duty cycle is calculated as the percentage of time the pulse is 'high' which is +5 Volts in this case.

$$\% \text{ Duty cycle} = \frac{t_p}{T} \tag{6.2}$$

The refresh frequency for the AROD hardware-in-the-loop test was chosen to be equal to the controller frequency of 40 Hertz. This gives a period T of 0.025 seconds or 25 milli-seconds. The minimum pulse width required for the servos is approximately 0.3 milli-seconds so:

$$\text{Min. \% Duty Cycle} = \frac{0.3}{25} = 0.012 \tag{6.3}$$

This corresponds to a vane deflection of -100 degrees. The maximum pulse width required for a $+100$ degree deflection is approximately 2.4 milli-seconds so:

$$\text{Max. \% Duty Cycle} = \frac{2.4}{25} = 0.096 \tag{6.4}$$

The pulse width corresponding to a centered position, or zero degrees of deflection, is approximately 1.05 milli-seconds so:

$$\text{Centered \% Duty Cycle} = \frac{1.05}{25} = 0.041 \tag{6.5}$$

Combining these gives:

$$\text{\% Duty Cycle} = 0.00041 \cdot (\text{Desired deflection in degrees}) + 0.053 \tag{6.6}$$

The algebraic block 'degrees_to_PWM' included in each of the four vane SuperBlocks and shown in Figure B.10 implements Equation 6.6.

3. Sensor Calibration

Before the sensors can be used reliably by the controller, they must be calibrated. Due to small changes in the operating voltages of the power supply, calibration is required each time the controller is started. For the original hardware-in-the-loop test, a separate C code program was run to calibrate the actuator sensors. Each time the controller is started the I/O devices must be initialized. This results in small changes in the measured voltages on the analog to digital I/O device from

one initialization to the next. For this reason the 'chained' IA picture was used so that the I/O would not have to be re-initialized after calibration. The procedure for

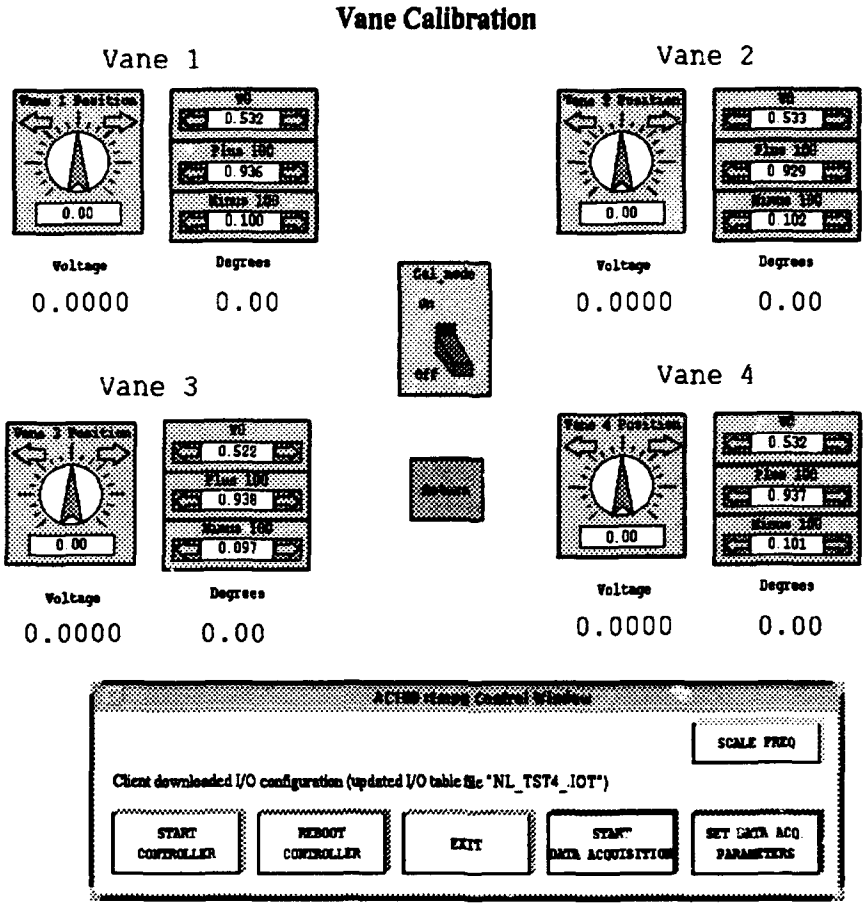


Figure 6.8: Interactive Animation Calibration Screen

calibration involves measuring the sensed voltage for vane positions of -100 deg, +100 deg, and zero degrees. The model uses these measured voltages in the equation used to convert the measured sensor voltage into the correct vane position in degrees:

$$\text{Vane position} = (V_{meas} - V_0) \times \frac{200}{V_{+100} - V_{-100}} \tag{6.7}$$

After starting the controller, the calibration routine is completed as follows:

- The user clicks on the 'calibrate' button to bring up the calibration screen.

- Next the user clicks on the 'Cal_mode' switch which connects the calibration inputs to the actuators.
- The voltage for 0 degrees is then entered in the 'V0' input for each vane.
- Next the position inputs are all changed to +100 degrees and the displayed voltage is input with the 'Plus 100' bar for each vane.
- Finally, the position inputs are all changed to -100 degrees and the 'Minus 100' inputs are changed accordingly.

The user can then switch the 'Cal_Mode' switch back to 'off' and click on the 'Return' button.

4. Data Acquisition Editor

A very useful feature of the AC100 is real–time data acquisition. The user can record any or all of the inputs and outputs to a C30 project. In this way, the user can get a very detailed analysis of the performance of a particular model and/or controller. The user first selects 'Data Acquisition Editor' from the AC100 GUI, Figure 6.4. The user will be presented with the screen shown in Figure 6.9. To record an input, simply select 'ON' under the 'setting' column. If the 'decimation_factor' is left at '1' then the value of that input will be recorded every time step. To select an output, toggle the 'Display' selector at the bottom of the screen from 'SB_INPUTS' to 'SB_OUTPUTS'. When all of the desired inputs and outputs have been selected, click 'DONE'. The AC100 GUI will return.

Once the user selects 'Download and Run', the 'AC100 rtmpg Control Window' and the Interactive Animation display, Figure 6.5, will appear. To start recording data, the user selects 'START DATA ACQUISITION'. This will create a file in

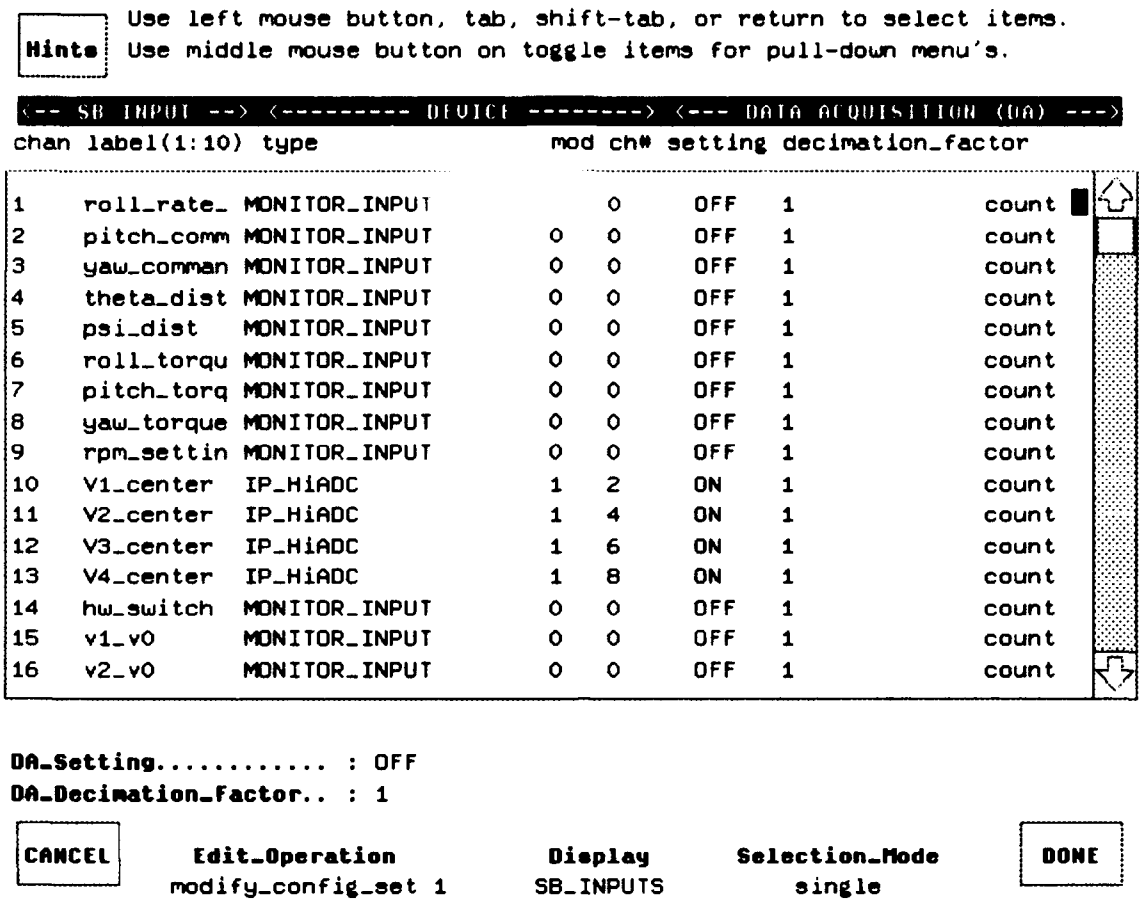


Figure 6.9: Data Acquisition Editor

the project directory with the project name and '_1.raw' appended. The number will be automatically incremented so that many data files may be collected. The button will also change to 'STOP DATA ACQUISITION'. If data acquisition is started before 'Start Controller', the data acquisition will begin with the first time step of the controller. Data acquisition stops automatically if the controller is stopped or rebooted.

After selecting 'REBOOT CONTROLLER', the user is returned to the AC100 GUI. The user then selects 'Convert raw data'. This will create a file with the same name as the '.raw' file with '.dat' as the file extension. This data file can

then be loaded directly into $MATRIX_X$. The $MATRIX_X$ variable names will be the same as those used as the input or output names in the SystemBuild model. These variables will be vectors with lengths depending on the amount of time that data was recorded. Plots of these variables are made the same as for any $MATRIX_X$ variable.

C. CONNECTING THE HARDWARE

Now that all of the software tools have been developed, the hardware must be physically connected to the control computer. Figure 6.10 shows the complete setup for the hardware-in-the-loop test using the AC100 Model C30. The SPARC workstation is the user's main interface with the controller. The Interactive Animation picture is updated via the ethernet connection with the C30. The C30 runs the controller, the aircraft model, and interfaces with all of the hardware.

The wiring harness was developed so that the first test flight of AROD could be done with the control computer remaining on the ground and connected to the AROD with a tether. The wiring harness and tether are designed to supply power to the AROD and to provide all of the control signals and return all of the sensor outputs. Figure 6.11 shows the connector end of the tether. For this test, only the 5 volt power lines and the vane signals were used.

The servo motors have two sets of wires emerging from the case. Each set contains a red, white, and black wire. The wires attached to the narrow side of the servos are the input wires and the wires attached to the wide side are the sensor outputs. Table 6.2 lists the function of each of these wires.

The pin diagrams for each of the C30 Modules are in [Ref. 22]. Figure 6.12 shows the complete wiring for the AROD hardware-in-the-loop test on the C30. The wiring harness box is a PC shell containing screw terminal blocks and a PC power

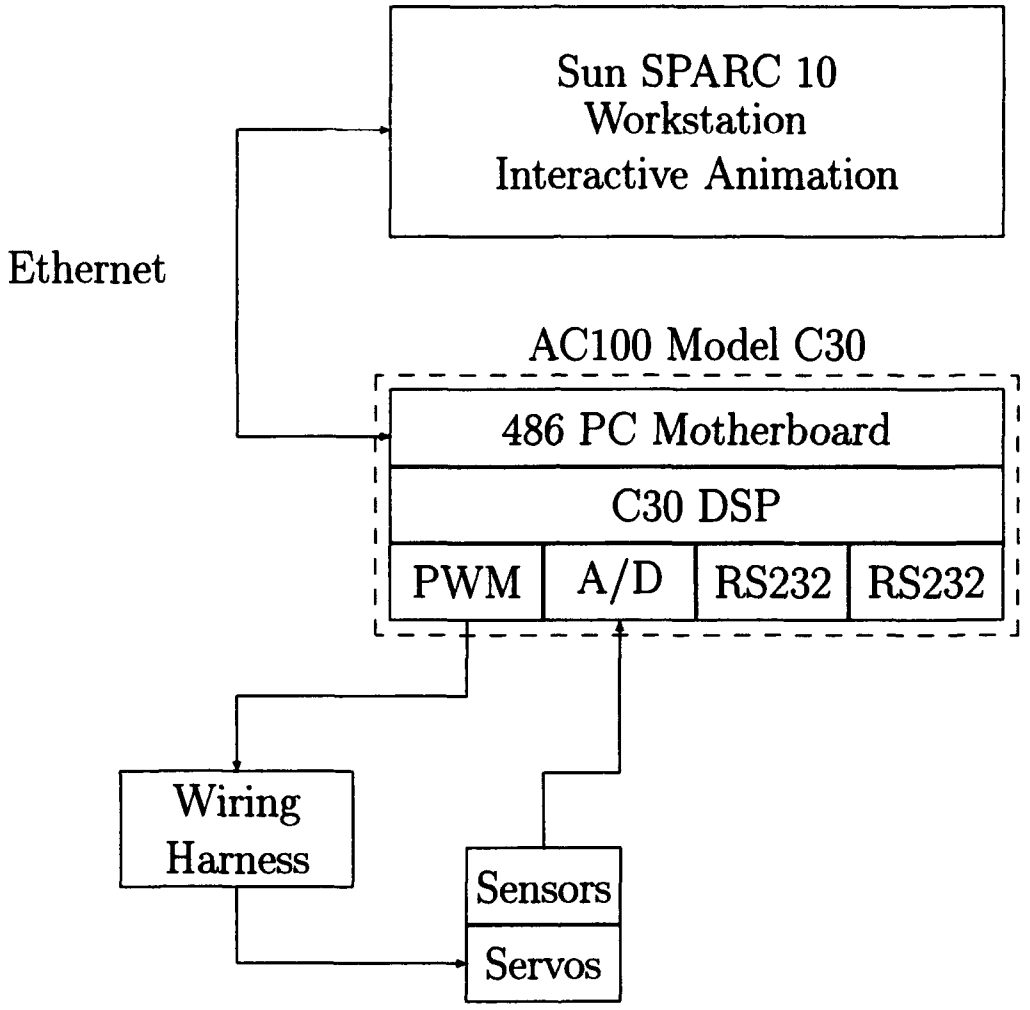
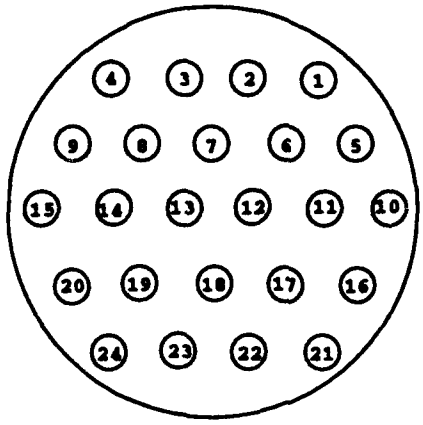


Figure 6.10: AC100 Model C30 Hardware–In–The–Loop Setup

supply as well as a 24 Volt power supply. The wire connections internal to the wiring harness are not shown.



1. Vane #1 Signal
2. Vane #2 Signal
3. Vane #3 Signal
4. Vane #4 Signal
5. Throttle Signal
6. Pitch Signal
7. Yaw Signal
8. Kill Switch
9. Kill Switch
10. Tachometer
11. +5 Volt Power
12. 5 Volt Ground
13. +24 Volt Power
14. 24 Volt Ground
15. Sheild
16. Vane #1 Sensor
17. Vane #2 Sensor
18. Vane #3 Sensor
19. Vane #4 Sensor
20.
21. Roll Rate Sensor
22. Pitch Rate Sensor
23. Yaw Rate Sensor
24.

Figure 6.11: Connector on end of Wiring Harness Tether

TABLE 6.2: SERVO MOTOR WIRING

	Control Inputs	Sensor Outputs
Red	+5 Volts	High Tap
White	PWM Input	Center Tap
Black	Ground	Ground

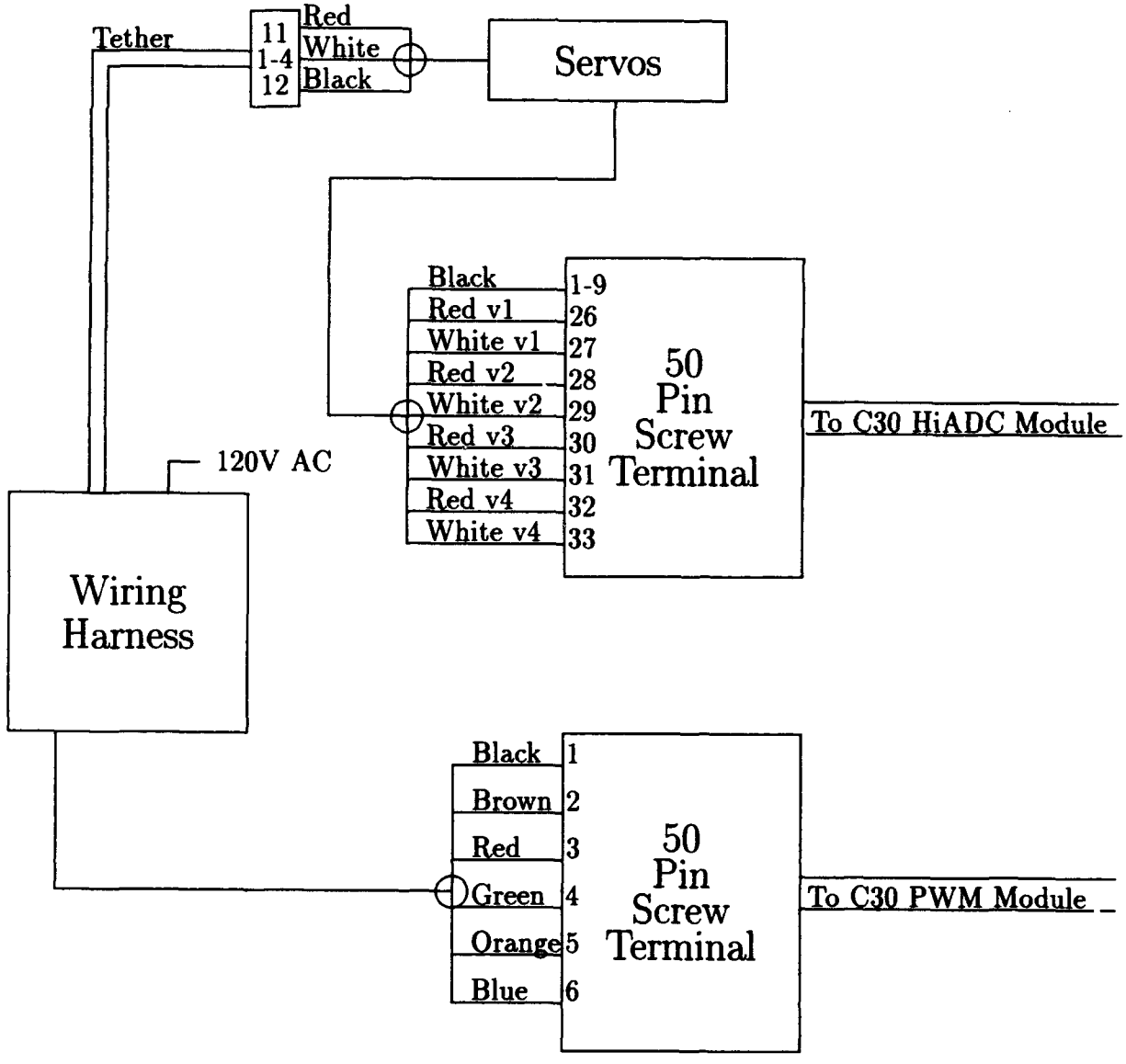


Figure 6.12: AC100 Model C30 Hardware-In-The-Loop Test Wiring Diagram

VII. RESULTS, CONCLUSIONS, AND RECOMMENDATIONS

A. HARDWARE–IN–THE–LOOP TEST RESULTS

The hardware-in-the-loop test of N. Sivashankar's controller showed the controller to be unstable. Analyzing the actuators as discussed in Chapter V revealed that the controller was changing the commanded position of the vanes faster than the vanes could respond. The controller gain was re-computed using the original synthesis model and the H_∞ theory procedure outlined in Chapter IV Section A. The cost function weighting matrices were adjusted to reduce the control loop bandwidth to less than 2 radians to account for the limited performance of the actuators. The resulting bandwidth for the control and command loops are compared to the originals in Figure 7.1. The first three sub-plots show the old (solid) and new (dashed) control loop bandwidth. The second three sub-plots show the old and new command loop bandwidth.

The new control gain was then entered into the controller and tested. The hardware-in-the-loop test of the new controller was successful and showed slower responses to disturbances as expected. For a comparison, the new controller was subjected to the same series of disturbances as the original controller. Figure 7.2 shows the SystemBuild disturbance rejection plot from the original controller. The hardware-in-the-loop disturbance rejection plot is similar with some noise [Ref. 3] but was not available for reproduction in this report. The disturbances introduced for all of these tests are listed in Table 7.1. Figure 7.3 shows the pitch and yaw errors

recorded from the hardware-in-the-loop test of the new controller. Notice that there is relatively little noise in the new controller due to the new sensor design.

TABLE 7.1: PITCH AND YAW DISTURBANCES IN RADIANS

Time (seconds)	0	4	9	15
Pitch Disturbance	0.1	0.2	0	0.2
Yaw Disturbance	-0.1	0	-0.2	0.2

B. CONCLUSIONS

Based on the data presented in this thesis the following conclusions are drawn:

- Automation using the AC100 Model C30 dramatically improves the time to first prototype. Valuable time is saved by not having to write code for the control computer and the hardware I/O drivers. The user only needs a basic understanding of the hardware and it's requirements. All required code is generated automatically by the AC100 software.

- Improvements to the model or controller can be implemented and tested immediately. For the same reasons as above, any changes made at the block diagram level can be tested immediately with a few mouse clicks.

- Real-time data acquisition allows for detailed analysis of test results. Since all of the inputs and outputs can be recorded at each time step, the data can be scrutinized thoroughly after a test. This is a tremendous help when trying to find errors created by improper timing.

- Hardware-in-the-loop testing does not fully validate a controller if the test is not performed real-time. The original controller was considered stable after initial hardware-in-the-loop testing, but was not stable when tested real-time.

C. RECOMMENDATIONS

Considering the conclusions introduced above and the experience gained while conducting this thesis, the following recommendations are made:

- Investigate sending IA data to additional ethernet address and capturing data for Virtual Prototying on Designer's Workbench. Previous thesis work has demonstrated the extraordinary benefits of Virtual Prototyping. Presently the data from an AC100 Model C30 hardware-in-the-loop test would have to be recorded and then moved to another file for use by Designer's Workbench. Sending data directly from the AC100 Model C30 to Designer's Workbench would allow real-time 3-Dimensional representation of the hardware-in-the-loop test data.

- Investigate using graphics programs with C30 for field display of data. Currently the AC100 Model C30 sends all of the test data to the workstation for display. The AC100 Model C30 would be very useful for a tethered flight test of the AROD, however, this would currently require a portable workstation to display the data from the AC100 Model C30. The data could easily be displayed on the AC100 Model C30 display with the use of a DOS or Windows graphics interface.

- Incorporate the AC100 Model C30 into the avionics design course work. Valuable experience is gained when testing a controller with hardware-in-the-loop . Students learn to consider all of the requirements for interfacing such as:

  - signal conversion (i.e., analog to digital, volts to degrees, degrees to PWM, etc.)

- power requirements
- timing (when control signals are generated vice when the sensor output is available)
- interference (interaction of control signals, Radio Frequency interference)
- data bandwidth (information required verses information available

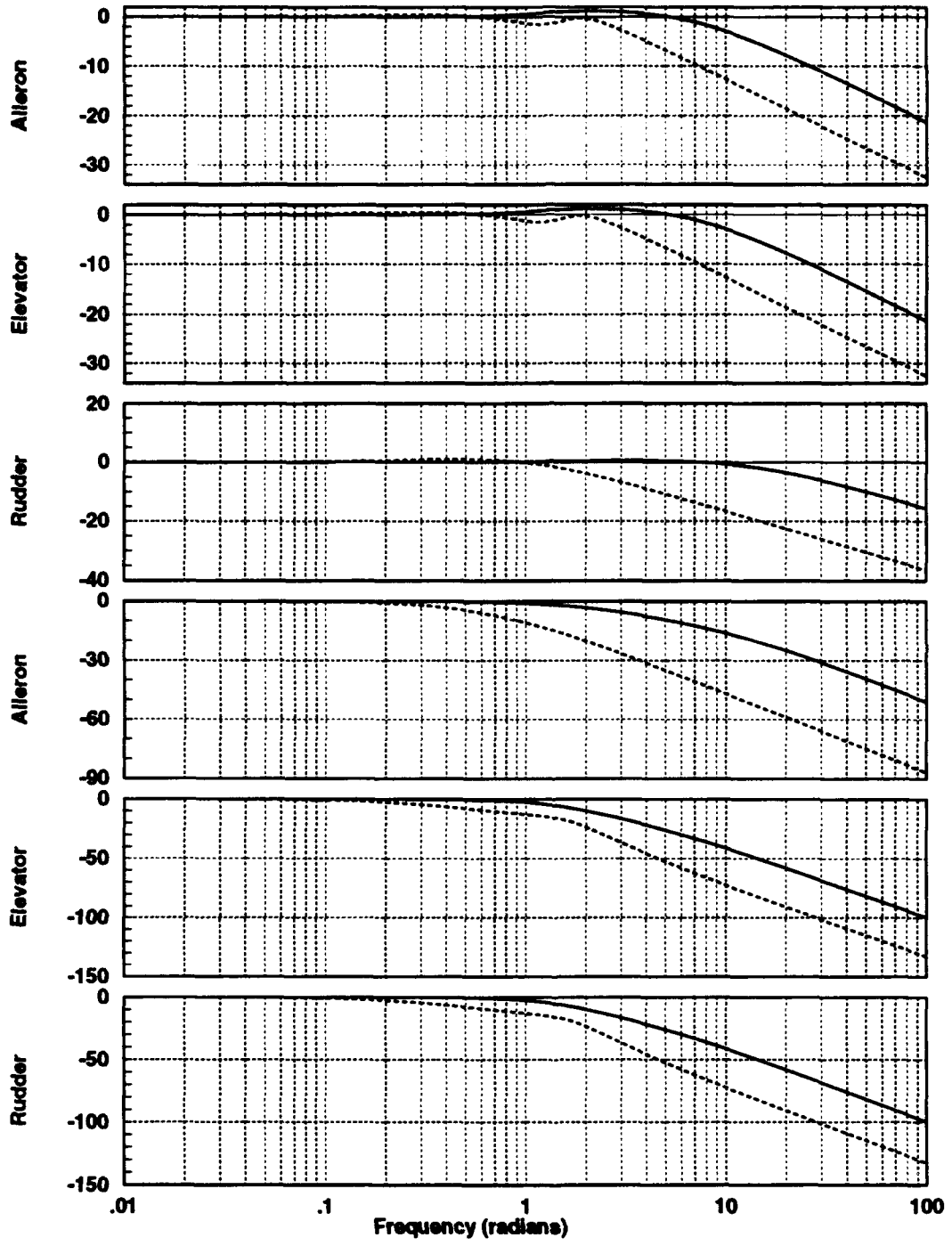


Figure 7.1: Comparison of Bandwidth for Controllers

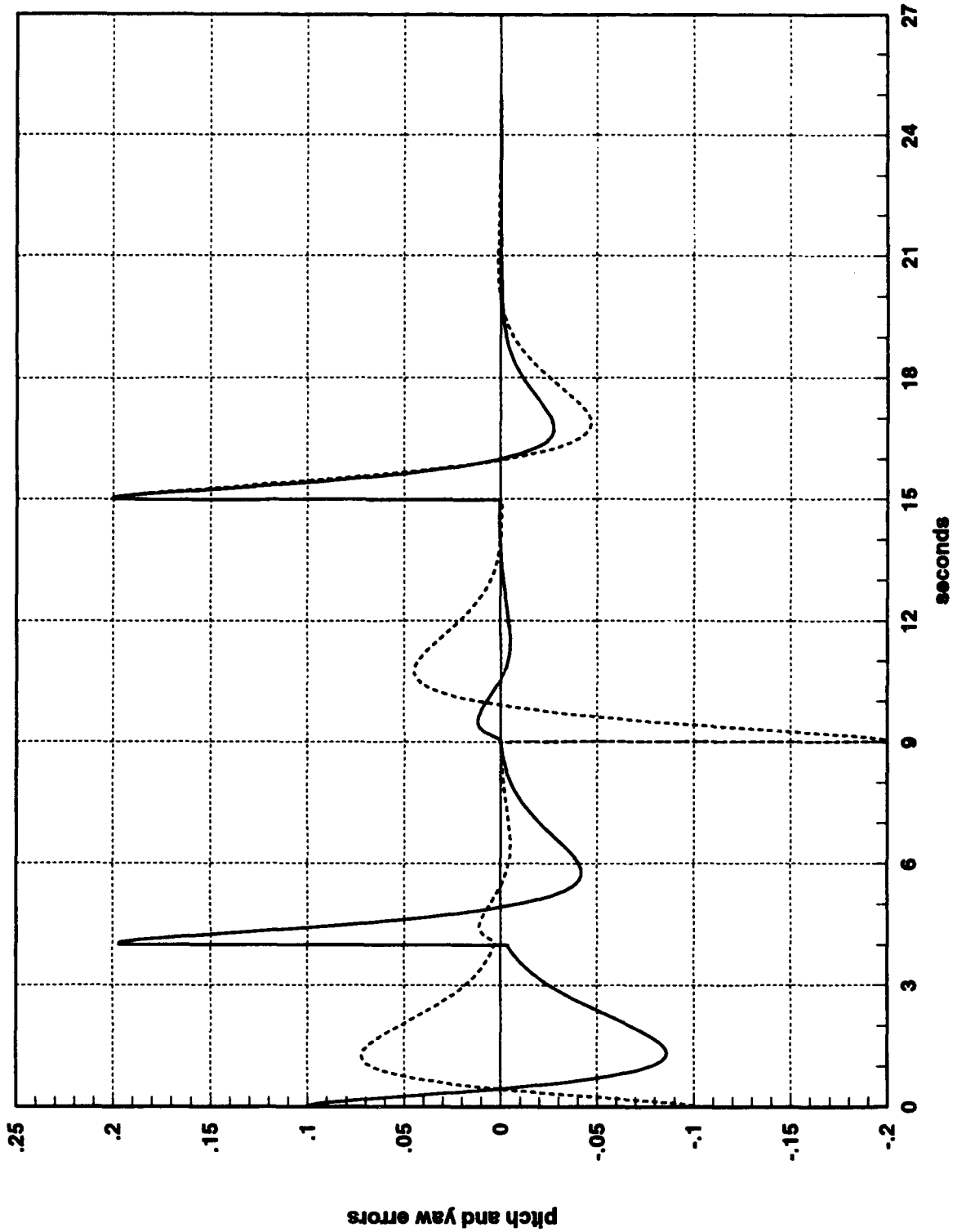


Figure 7.2: Disturbance Rejection for Old Controller

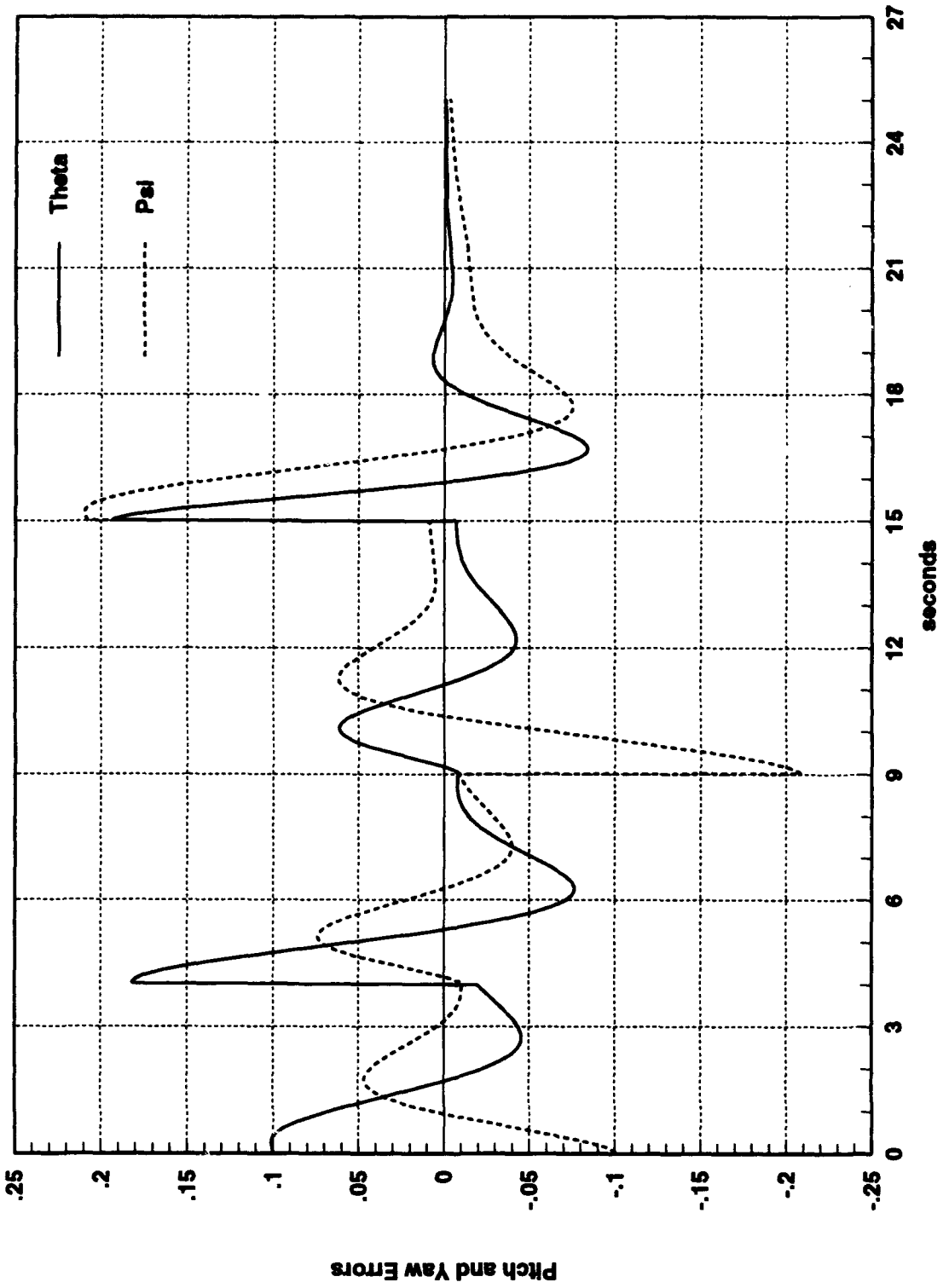


Figure 7.3: Disturbance Rejection for New Controller

APPENDIX A

MATLAB FILES

A. GET_XDOT.M

```
function    [ x_dot] = get_xdot(invect)
%
% Function computes x_dot given an input vector which is x and u_c
% muxed together. u_c is first four and x is last nine. The inputs
% order is elevator, rudder, aileron, rpm_setting. The states are
% u, v, w, p, q, r, phi, theta, psi.
%
%%%%%%%%          First step is to demux the input vector

u_c=invect(1:4);
%    u_c(1)=delta e    (elevator)
%    u_c(2)=delta r    (rudder)
%    u_c(3)=delta a    (aileron)
drpm = u_c(4);        %(throttle)

v=invect(5:7);
%    v(1)=u    (x velocity)
%    v(2)=v    (y velocity)
%    v(3)=w    (z velocity)

omega=invect(8:10);
```

```
p=omega(1); %(roll rate)
q=omega(2); %(pitch rate)
r=omega(3); %(yaw rate)

lambda=invect(11:13);
phi=lambda(1);      %(bank angle)
theta=lambda(2); %(pitch angle)
psi=lambda(3);      %(yaw angle)

x=[ v; omega; lambda; ] ;

%%%%%%%%          Next get constant values for:
% m,Ib,Ir,S,Cfo,dCfdx,dCfdd,M1,rho,Ar,g
%
constval;
% constval is a Matlab script file, not a function, and sets the
% values in the Matlab environment for use by all functions.

%%%%%%%%          Form omega cross matrix and compute Vt and q
wx=[ 0 -r q; r 0 -p; -q p 0 ] ;
Vt=sqrt(v(1)^ 2+v(2)^ ?+v(3)^ 2);
qbar=.5* rho* Vt^ 2;

%%%%%%%%          Form sine and cosine abreviations
%
ch=cos(phi);
sh=sin(phi);
```

```
ct=cos(theta);
st=sin(theta);
cs=cos(psi);
ss=sin(psi);

%%%%%%%%          Form force due to gravity
%             B       U
%               R *    F      = force due to gravity, moment = 0
%             U          g
%
%   2-3-1 rotation
%
RubFg=[ -cs* st* g
          (ch* ss* st+sh* ct)* g
          (ch* ct-sh* ss* st)* g    ] ;

%%%%%%%%          Form Rwb (transform wind coordinates to body
alpha=asin(v(2));
beta=-asin(v(3));
Rwb=[ cos(alpha)* cos(beta) -cos(alpha)* sin(beta) -sin(alpha)
                sin(beta)                    cos(beta)              0
       sin(alpha)* cos(beta) -sin(alpha)* sin(beta)    cos(alpha) ] ;

%%%%%%%%          System is of the form
%             .
%             x = A x + B u + C          where u is the control inputs and
```

```
%                                          C is a combination of gravity and
%                                          other influences
%
%%%%      Form "A" matrix
%
%%%% USING THRUST VS RPM FROM BOB STONEY'S TEST RUNS
%
T=0.0297* drpm-104.7;
%
Vi=sqrt(T/2/Ar/rho);
%
% NOTE: derivatives are non-dimentionalized with qi (induced velocity)
% so add u to the induced velocity for total q
%
qt=.5* rho* (v(1)+Vi)^ 2;
%
It=Ib+Ir;
O=zeros(3,3);
I=eye(3);
%
% The generic 'A'matrix would be:
% A=([ -wx O; O -It\ (wx* It)] + [ (I/m)* Rwb O; O It\ Rwb ] * q* S* dCfdx*
M1);
%
A=[ -wx O; O -It\ (wx* It)] ;
```

```
%
%%%%      Form "B" matrix
%
% Note: control surface derivatives are non-dimentionalized using
% characteristic lengths from rotor
%
B1=[ (I/m)* Rwb O; O It\ I ] * qt* Sd* dCfdd;
%
%%%%% LT relates the duct swirl to the moment l produced by thrust
%
LT=-0.0542* T-0.9138;
wr=[ drpm* 2* pi/60;0;0] ;
%
B2=[ T/m; 0; 0; -It\ ([ LT; 0; 0;] +(wx* Ir* wr)) ] ;
%
%%%%      Form "C" matrix
%
% Generic 'C'matrix would be
% C=[ (I/m)* RubFg; 0;0;0; ] +[ (I/m)* Rwb O; O It\ Rwb ] * qbar* S* Cfo;
%
C=[ RubFg; 0;0;0; ] ;
%
%%%%      Form Drag matrix
%
% Note: this is not aerodynamic drag, this is an estimate of the
```

```
% parasitic drag elements.
%
sb=6.7708;
s=S(2,2);
su=sign(v(1));
sv=sign(v(2));
sw=sign(v(3));
sp=sign(p);
sq=sign(q);
sr=sign(r);
Sdrag=diag([ su* Ar,sv* sb,sw* (s+sb),sp* .5* s,sq* .7* (s+sb),sr* .5* sb] );
Vm=[ (v.^ 2)/m; It\ (omega.^ 2) ] ;
D=rho* diag(Cfo)* Sdrag* Vm;

%%%%%%%%      Form lambda dot using 2-3-1 rotation
%
ldot=[ p-q* ch* ss/cs+r* sh* ss/cs
            q* ch/cs-r* sh/cs
            q* sh+r* ch                    ] ;

%%%%%%%%      Form totals
%
vw_dot=A* [ v; omega ] +B1* u_c(1:3)+B2+C+D;
x_dot=[ vw_dot; ldot; ] ;
```

B. CONSTVAL.M

```
%
% [ m,Ib,Ir,S,Sd,Cfo,dCfdx,dCfdd,M1,rho,Ar,g]
%
% This file returns all of the constants for the arod EOM
% Hover condition =) all aero derivatives are zero

% Total body mass
m=2.6419;    % slugs

% Body mass moment of inertia
Ib=[ 1.2312 0 0; 0 3.9548 0; 0 0 3.9825 ] ; %slug ft^ 2

% Prop mass moment of inertia
Ir=[ 0.00898 0 0; 0 0.0045 0; 0 0 0.0045 ] ; %slug ft^ 2

% Standard length matrix for wings
ss=9.4444;      bw=5.7;      cbarw=2.165;
S=diag([ -ss,ss,-ss,ss* bw,ss* cbarw,ss* bw] );

% Standard length matrix for control surfaces
sp=pi;      bp=1;      cbarp=1;
Sd=diag([ -sp,sp,-sp,sp* bp,sp* cbarp,sp* bp] );

% Cfo (used as a drag term in all three axes
Cfo=[ -.015; -.015; -.015; -0.015; -0.015; -0.015; ] ;

% dCf/d
dCfdx=0;
```

```
% dCf/dx_dot=0 for this model
% dCfdxd=[    ] ;

% dCf/ddelta
dCfdd=[ zeros(3,3); 0 0 1.438; -1.233 0 0; 0 -1.233 0 ] ;

% M1
Vt=1;
M1=diag([ 1/Vt,1/Vt,1/Vt,bw/(2* Vt),cbarw/(2* Vt),bw/(2* Vt)] );

% rho
rho=.002377;      %slugs/ft^ 3

% area
R=1;
Ar=pi* R;

% gravity
g=32.174;      %ft/sec^ 2
```

APPENDIX B

MATRIX$_X$ BLOCK DIAGRAMS

A. AROD SystemBuild BLOCK DIAGRAMS

The AROD SystemBuild block diagram contains the following SuperBlocks:

- **actuators:** Figure B.1 shows the SuperBlock containing 4 actuator SuperBlocks, one for each vane.
- **actuator_1:** Figure B.2 shows the actuator model for the first vane and is identical to the other vane actuators.
- **actuator_2:** Not shown.
- **actuator_3:** Not shown.
- **actuator_4:** Not shown.
- **ang_velocity_eq:** Figure B.3 shows the SuperBlock for the angular velocity equations. The values for the body and rotor inertia matrices are listed in Table 2.1.
- **dcont_wind:** Figure B.4 shows the controller SuperBlock for the AROD. The saturation block limits the throw of the vanes to ± 15 degrees. The two algebraic blocks convert the command signals to vanes signals in degrees, and the vane signals back to command signals in radians respectively.
- **filters:** Figure B.5 shows the four anti-aliasing filters discussed in Chapter V Section E. The values for the four vane filters are given in Equation 5.8.

- **integ_ang_vel:** Not shown.
- **integ_lin_vel:** Not shown.
- **Integ_sim:** This SuperBlock contains the SuperBlocks 'integ_ang_vel', 'integ_lin_vel', and 'int_ang_sim'. Each of these SuperBlocks contain three discrete integrators as shown in Figure 3.2 with the appropriate initial values. The 'int_ang_sim' SuperBlock also contains two sum blocks to add in the perturbations for θ and ψ.
- **int_ang_sim:** Not shown.
- **kinematics:** Figure B.6 shows the kinematics SuperBlock which contains the 'lin_velocity_eq', 'ang_velocity_eq', and 'L_dot_eq' SuperBlocks.
- **lin_velocity_eq:** Figure B.7 shows the equations for the linear forces acting on the aircraft. The 'T_value' SuperBlock contains Equation 3.1 and block 95 is the force due to gravity.
- **L_dot_eq:** Not shown. This SuperBlock is an implementation of Equation 2.58.
- **l_m_n_compute:** Figure B.8 computes the angular momentum terms. Blocks 5, 6, and 98 contain the appropriate stability derivatives and block 7 includes the moment due to propeller thrust given in Equation 3.2.
- **nl_tst4_hw:** Figure B.9 shows the top level SuperBlock for the hardware-in-the-loop test. The inputs and outputs from this SuperBlock are listed in the 'nl_tst4_hw.rtf' file and used in the Hardware Connection Editor.
- **T_value:** Not shown. This SuperBlock is Equation 3.1.

- **vane1:** Figure B.10 shows the Superblock for vane one and is identical to the other three SuperBlocks. Block 6 shows the conversion from degrees to percent duty cycle discussed in Section c of Chapter VI. Block 4 is the algebraic block discussed in Section 3 of Chapter VI.

- **vane2:** Not shown.

- **vane3:** Not shown.

- **vane4:** Not shown.

- **vane4x:** Figure B.11 shows the 'vane4x' SuperBlock which contains a SuperBlock for each of the four vanes and the 'filters' SuperBlock.

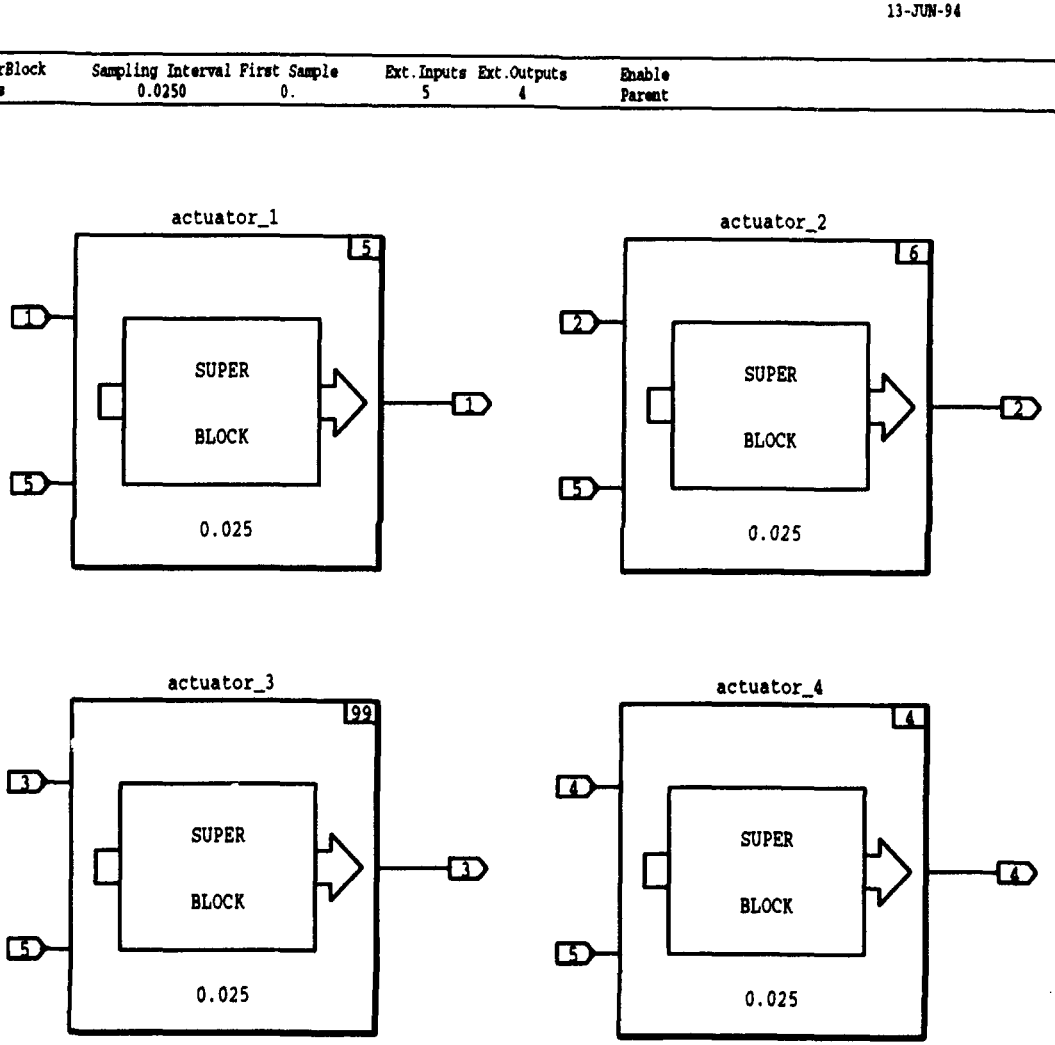


Figure B.1: Four Actuator Superblock

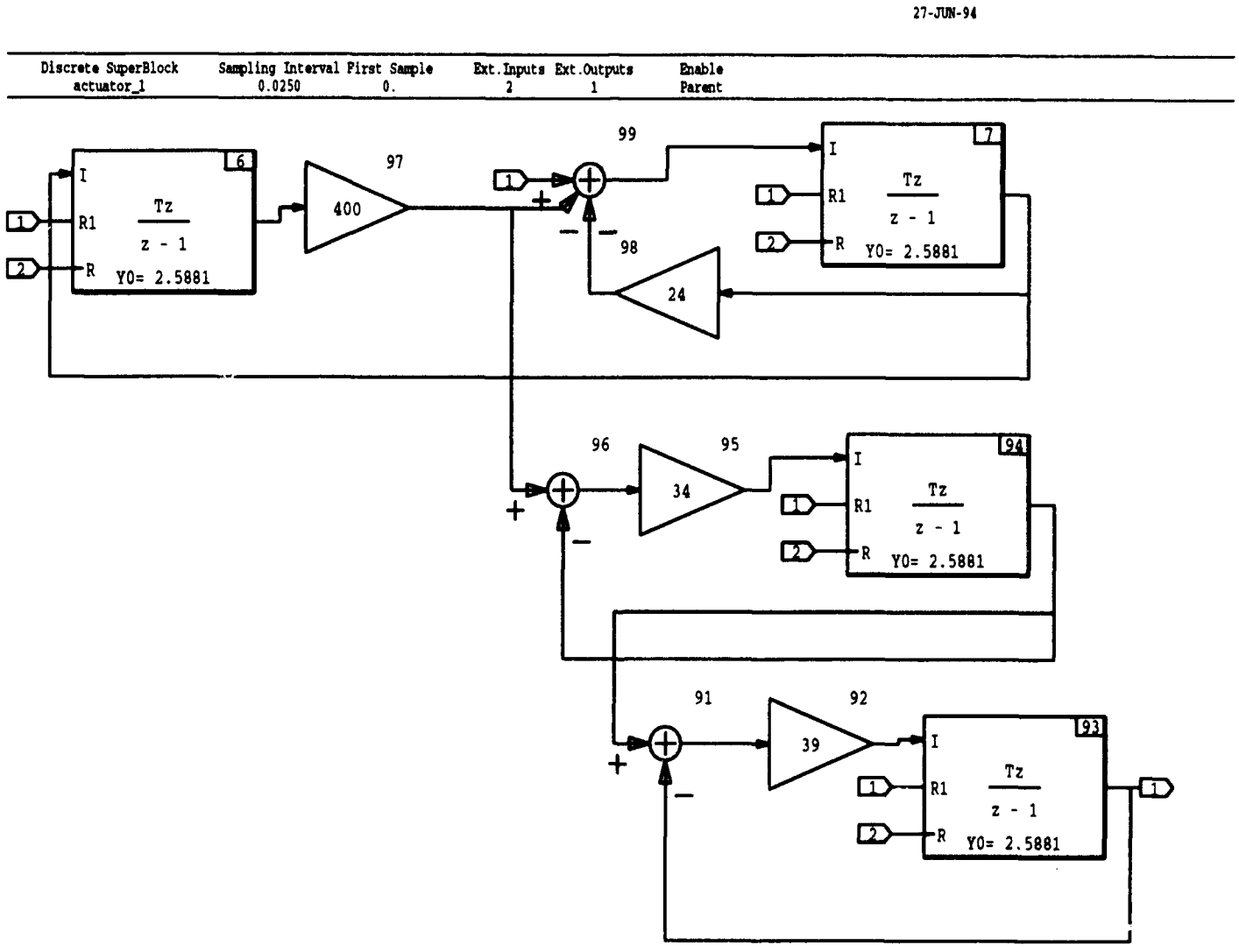


Figure B.2: Single Actuator Superblock

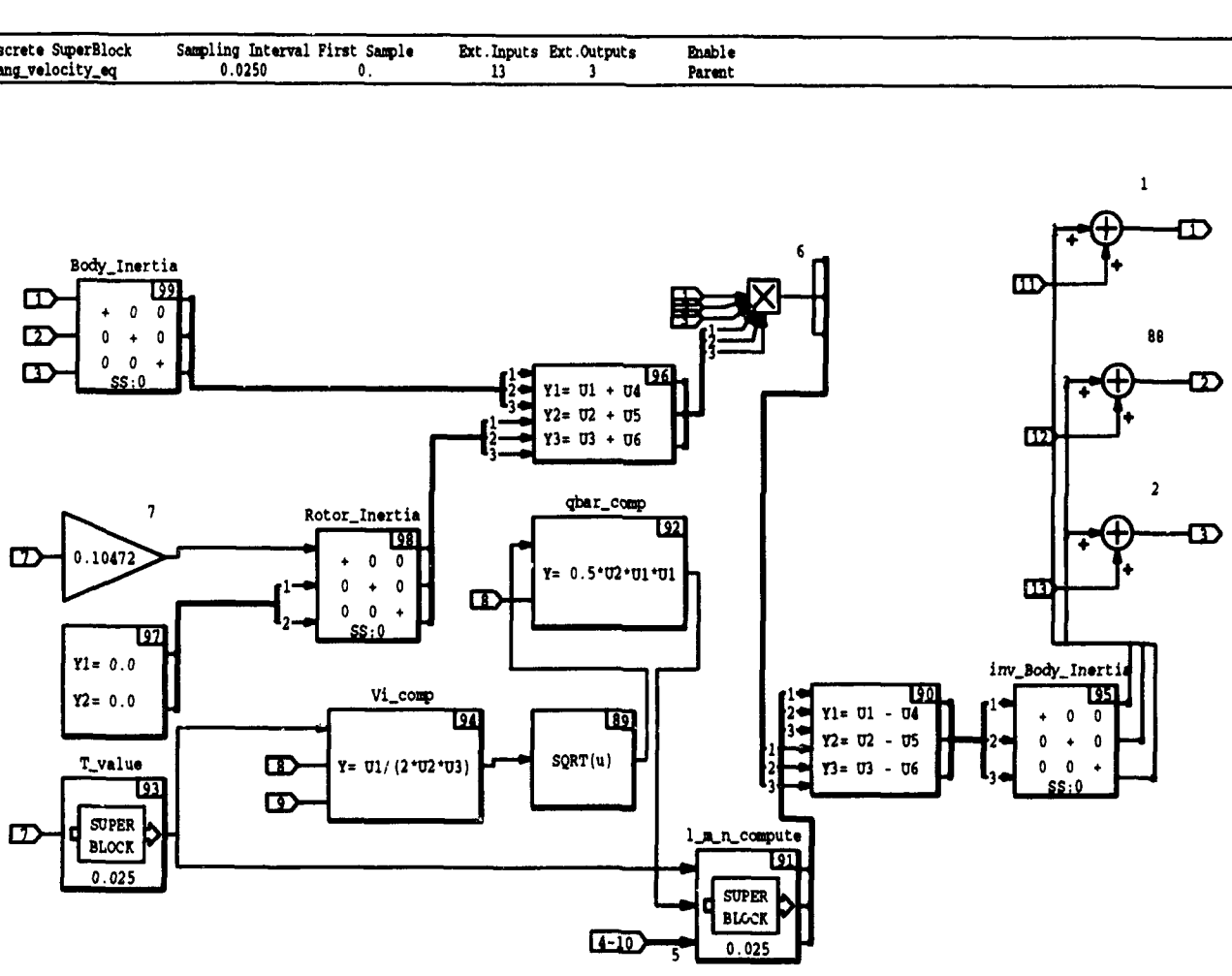


Figure B.3: Angular Velocity Equation Superblock

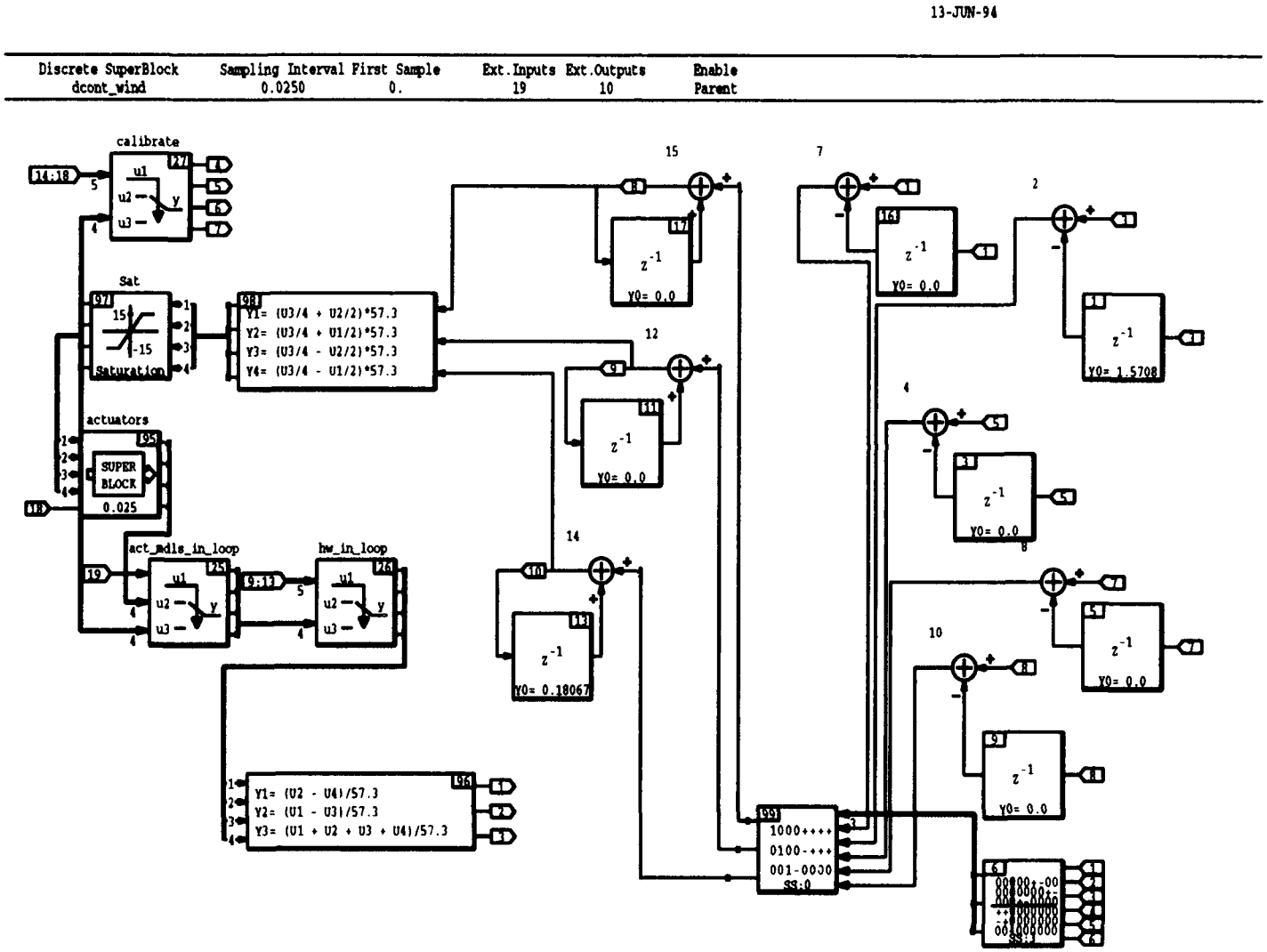


Figure B.4: Discrete Controller Superblock

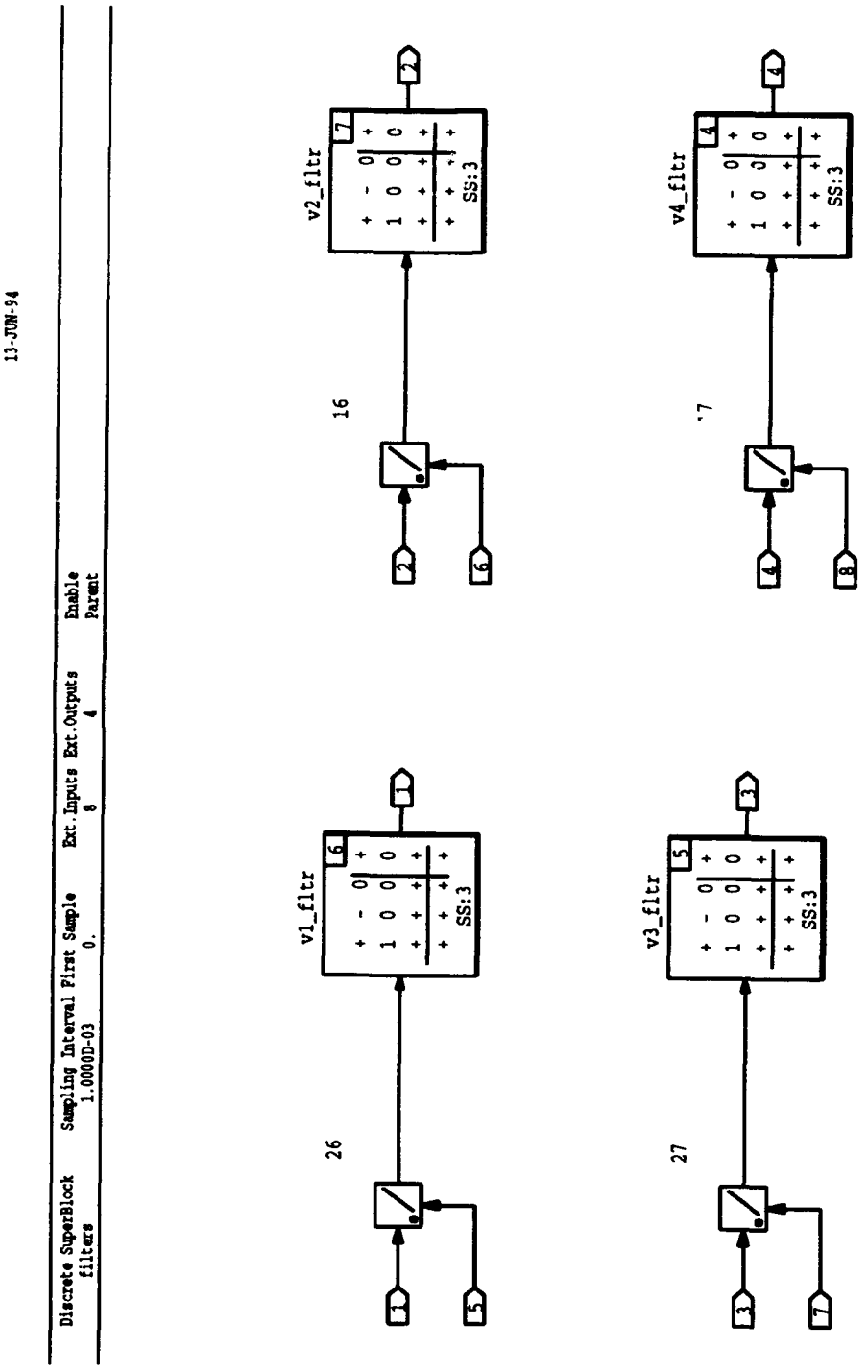


Figure B.5: Anti-Aliasing Filters Superblock

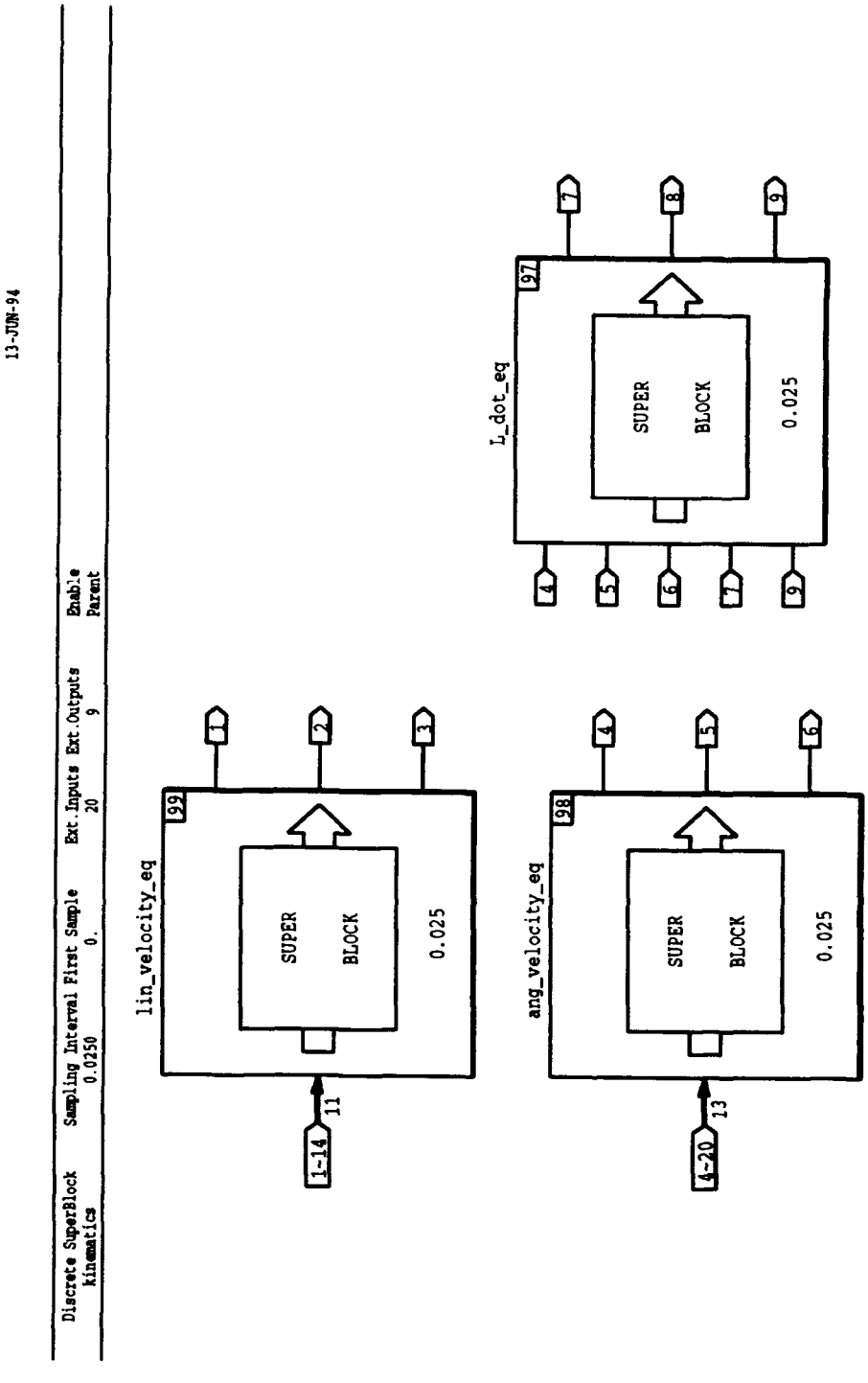


Figure B.6: AROD Kinematics Superblock

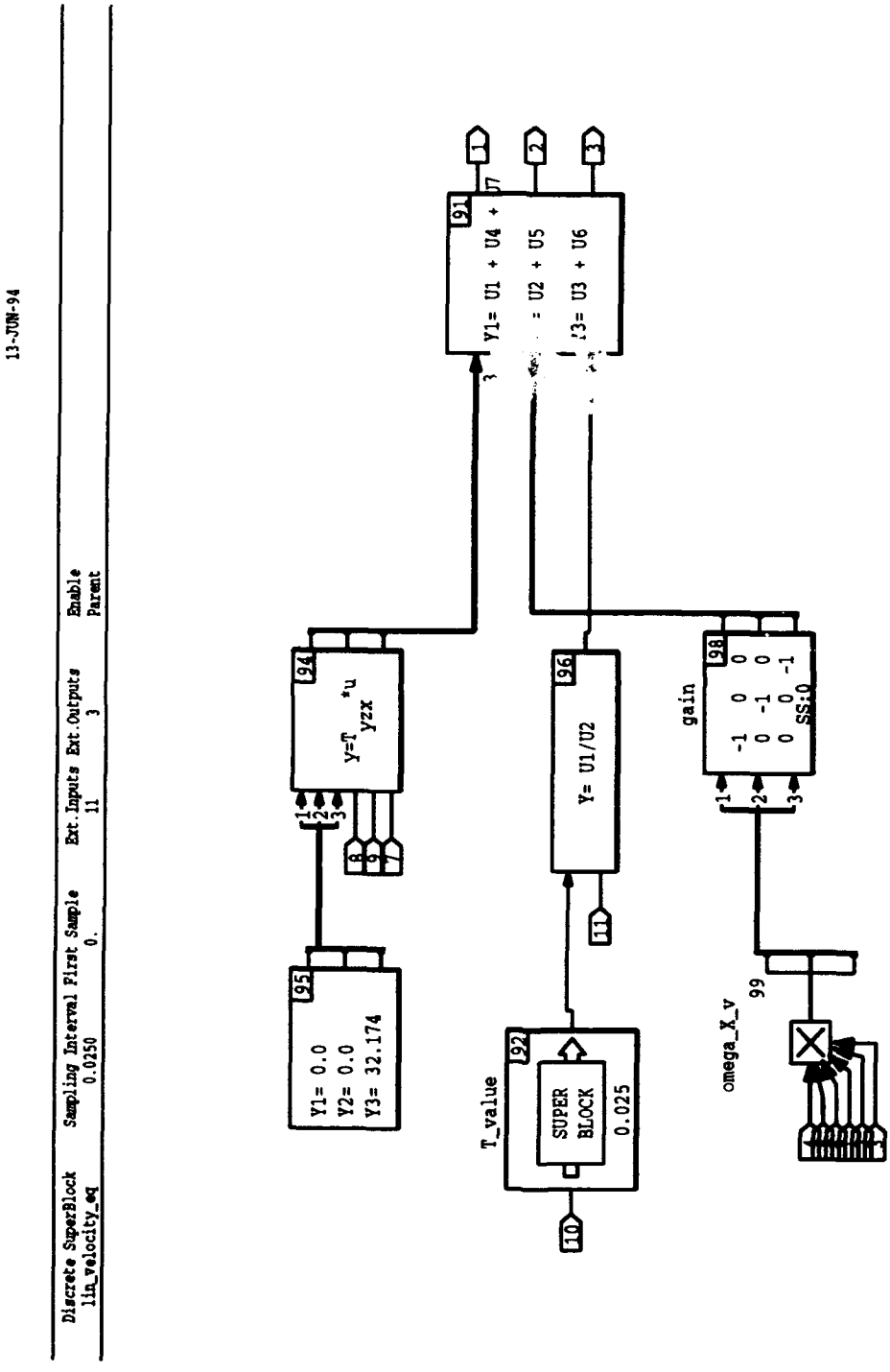


Figure B.7: Linear Velocity Equation Superblock

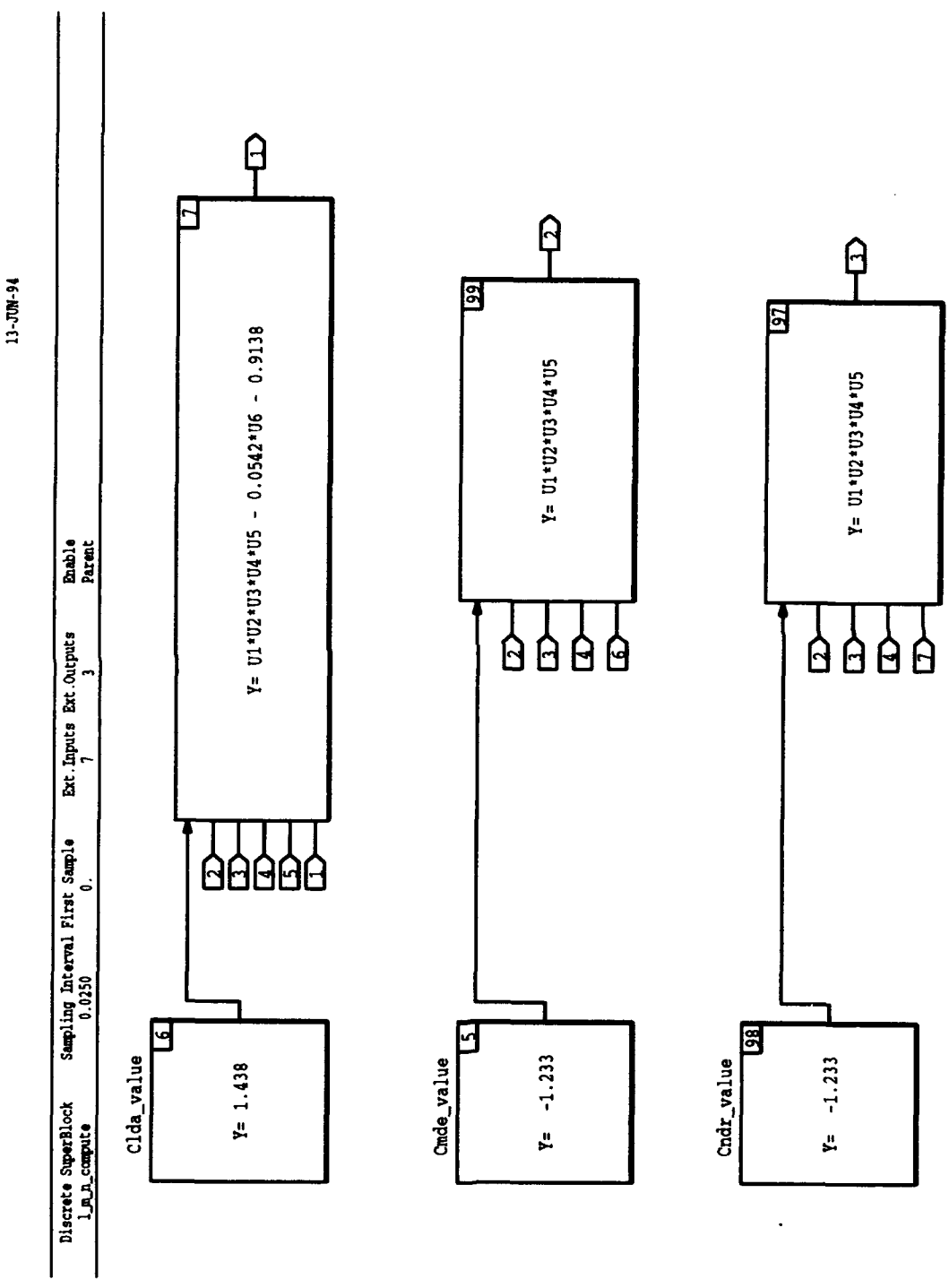


Figure B.8: Compute l,m, and n Superblock

Figure B.9: AROD Model and Controller Superblock

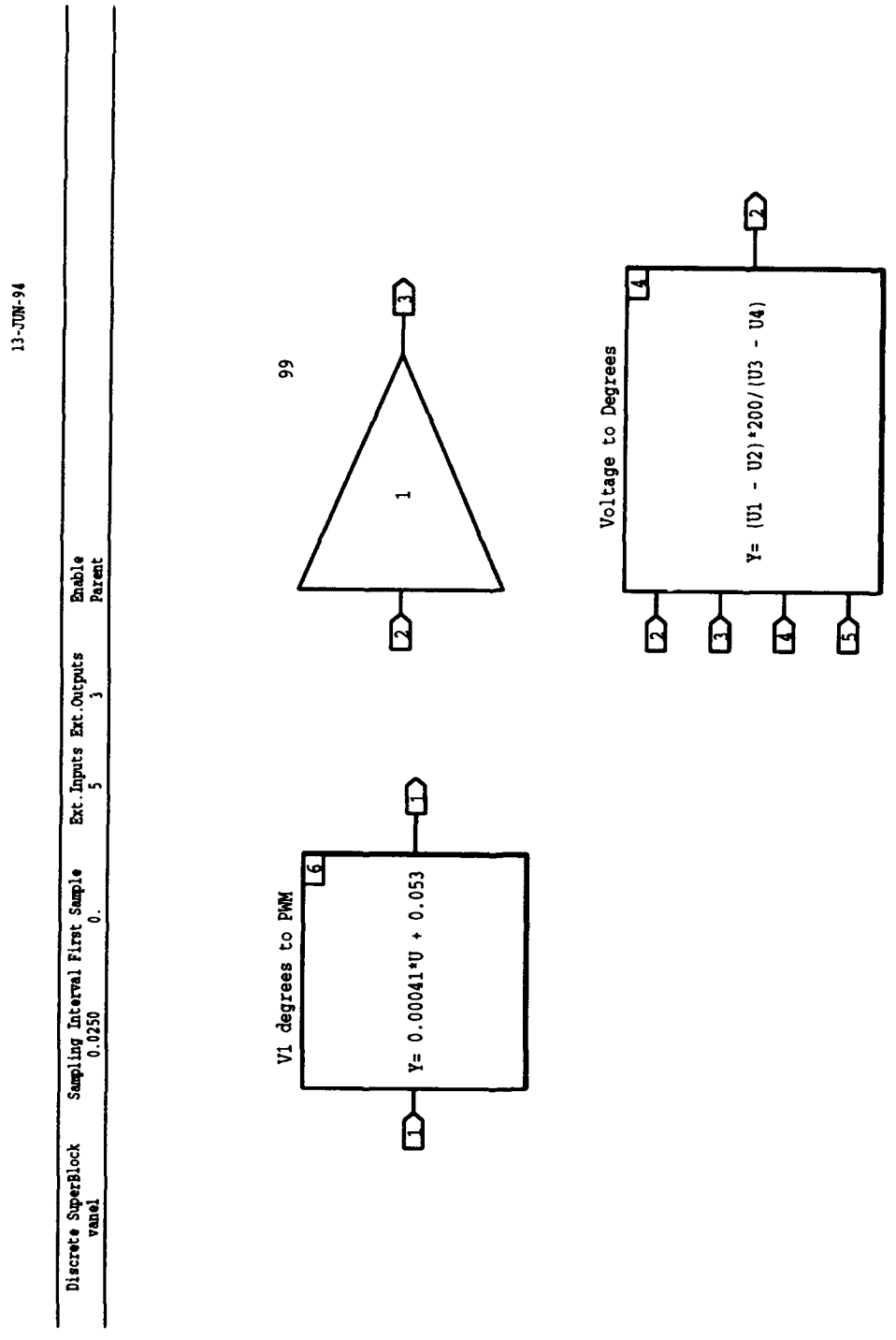


Figure B.10: Single Vane Superblock

13-JUN-94

Discrete SuperBlock	Sampling Interval	First Sample	Ext.Inputs	Ext.Outputs	Enable
vane4x	0.0250	0.	24	12	Parent

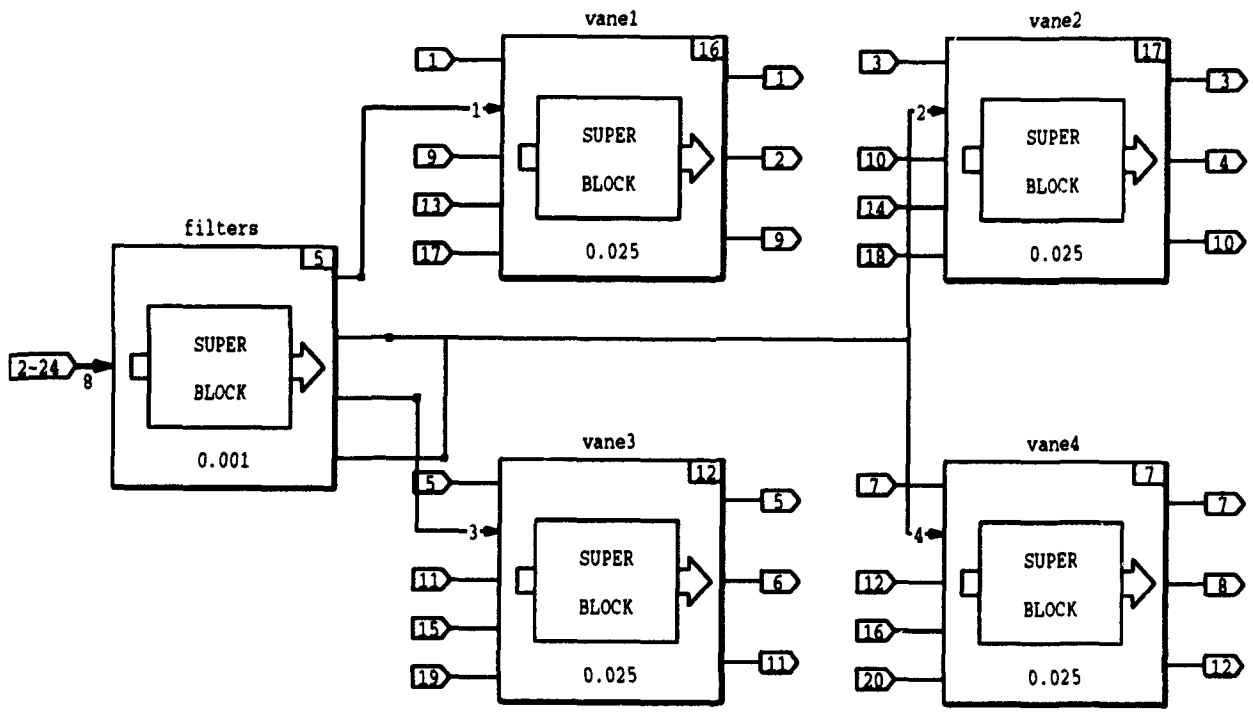


Figure B.11: Four Vane Superblock

LIST OF REFERENCES

[1] Hallberg, Eric N., "Design of a GPS Aided Guidance, Navigation, and Control System for Trajectory Control of an Air Vehicle," Master's Thesis, Department Of Aeronautics, Naval Postgraduate School, Monterey, CA, March 1994.

[2] Kuechenmeister, David R., "A Non-Linear Simulation For An Autonomous Unmanned Aerial Vehicle," Master's Thesis, Department Of Aeronautics, Naval Postgraduate School, Monterey, CA, September 1993.

[3] Sivashankar, N., "Design, Analysis and Hardware–in–the–Loop Testing of a Controller for the Unmanned Aerial Vehicle ARCHYTAS," Report on work done while a visiting scholar, Department Of Aeronautics, Naval Postgraduate School, Monterey, CA, August 1993.

[4] White, J.E., and Phelan, J.R., "Stability Augmentation for a Free Flying Ducted Fan," *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, Monterey, CA, Aug. 1987, pp 896–904.

[5] White, J.E., and Phelan, J.R., "Stability Augmentation and Control Decoupling for the Airborne Remotely Operated Device," *Journal of Guidance, Control, and Dynamics*, Vol. 14, No.1, 1991, pp 176-183.

[6] Junkins, J.L., *An Introduction to Dynamics and Control of Flexible Structures*, AIAA, Washington D.C., 1992.

[7] Schmidt, L.V., *Class Notes for AE3340*, U.S. Naval Postgraduate School, Monterey, CA. 1992.

[8] Greenwood, D.T., *Principles of Dynamics, 2nd Ed.*, Prentice–Hall, Englewood Cliffs, N.J., 1988.

[9] Roskam, J., *Airplane Flight Dynamics and Automatic Flight Controls*, Roskam Aviation and Engineering corp, Ottawa, KS, 1979

[10] Thompson, C.M., "Aircraft Equations of Motion and Forming Linear Models," Boeing Document D6–54972, Boeing Commercial Airplane Company, Seattle, Washington, 1989.

[11] Kaminer, I.I., *Class Notes for AA4276,* U.S. Naval Postgraduate School, Monterey, CA. 1994.

[12] Stoney, R.B., "Design, Fabrication, and Test of a Vertical Attitude Take–Off and Landing Unmanned Air Vehicle," *Engineer's Thesis, Department of Aeronautics*, Naval Postgraduate School, Monterey, CA, June 1993.

[13] Byerly, J., "Development of Equations-of-Motion in MATRIX$_X$/SystemBuild," Report on work completed for Office Course AE-4900, Department Of Aeronautics, Naval Postgraduate School, Monterey, CA, June 1994.

[14] Maciejowski, J.M., *Multivariable Feedback Design,* Addison–Wesley, Reading, Massachusetts, 1989.

[15] Collins, D.J., *Class Notes for AA4342,* U.S. Naval Postgraduate School, Monterey, CA. 1994.

[16] Ogata, K., *Modern Control Engineering,* Prentice–Hall, Englewood Cliffs, N.J., 1990.

[17] Oppenheim, A.V., Willsky, A.S., and Young, I.T., *Signals and Systems*, Prentice–Hall, Englewood Cliffs, N.J., 1983.

[18] Integrated Systems, Inc. *AC–100 Reference*, July 1993.

[19] Integrated Systems, Inc. *AC–100 User's Guide*, July 1993.

[20] Integrated Systems, Inc. *Autocode User's Guide*, August 1992.

[21] Integrated Systems, Inc. *SystemBuild User's Guide*, November 1991.

[22] Integrated Systems, Inc. *AC–100 Model C30 Supplemental Reference*, July 1993.

[23] Noyes, B., "Hardware in the Loop Testing with the AC100C30 and Flight Management Unit," Report on work completed for Office Course AE-4900, Department Of Aeronautics, Naval Postgraduate School, Monterey, CA, June 1994.

INITIAL DISTRIBUTION LIST

		No. of Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2.	Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Dr. Isaac I. Kaminer Department of Aeronautics and Astronautics, Code AA/KA Naval Postgraduate School Monterey, CA 93943-5000	5
4.	Dr. Richard M. Howard Department of Aeronautics and Astronautics, Code AA/Ho Naval Postgraduate School Monterey, CA 93943-5000	1
5.	Chairman Department of Aeronautics and Astronautics Naval Postgraduate School Monterey, CA 93943-5000	2
6.	Nels G. Moats 1221 Hathaway Dr Colorado Springs, CO 80915	1
7.	Gene Savell P.O. Box 96 Citrus Heights, CA 95610	1

No. of Copies

8. Michael L. Moats — 2
   1360 Josselyn Canyon Road # 19
   Monterey, CA 93940

9. innovative systems & technologies corporation — 2
   P.O. Box 274150
   Tampa, FL 33688-4150